PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 9/28, 19/00, 11/00 — A1

(11) International Publication Number: WO 95/23373

(43) International Publication Date: 31 August 1995 (31.08.95)

(21) International Application Number: PCT/US95/02229

(22) International Filing Date: 23 February 1995 (23.02.95)

(30) Priority Data:

08/203,191	28 February 1994 (28.02.94)	US
08/229,609	19 April 1994 (19.04.94)	US
08/377,610	25 January 1995 (25.01.95)	US

(71) Applicant: TELEFLEX INFORMATION SYSTEMS, INC. [US/US]; Suite 100, 7736 Mc Cloud Road, Greensboro, NC 27408 (US).

(72) Inventors: PETERS, Michael, S.; 2312 North Elm Street, Grennsboro, NC 27408 (US). HOLT, Clayton, Walter; 2205 New Garden Road, Grennsboro, NC 27408 (US). ARNOLD, David, J., Jr.; 361 Worth Street, Asheboro, NC 27203 (US).

(74) Agents: LOBSENZ, Charles, B. et al.; Baker & Botts, L.L.P., The Warner, 1299 Pennsylvania Avenue, N.W., Washington, DC 20004 (US).

(81) Designated States: AM, AT, AU, BB, BG, BR, BY, CA, CH, CN, CZ, DE, DK, EE, ES, FI, GB, GE, HU, JP, KE, KG, KP, KR, KZ, LK, LR, LT, LU, LV, MD, MG, MN, MW, MX, NL, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TT, UA, UG, UZ, VN, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG), ARIPO patent (KE, MW, SD, SZ, UG).

Published

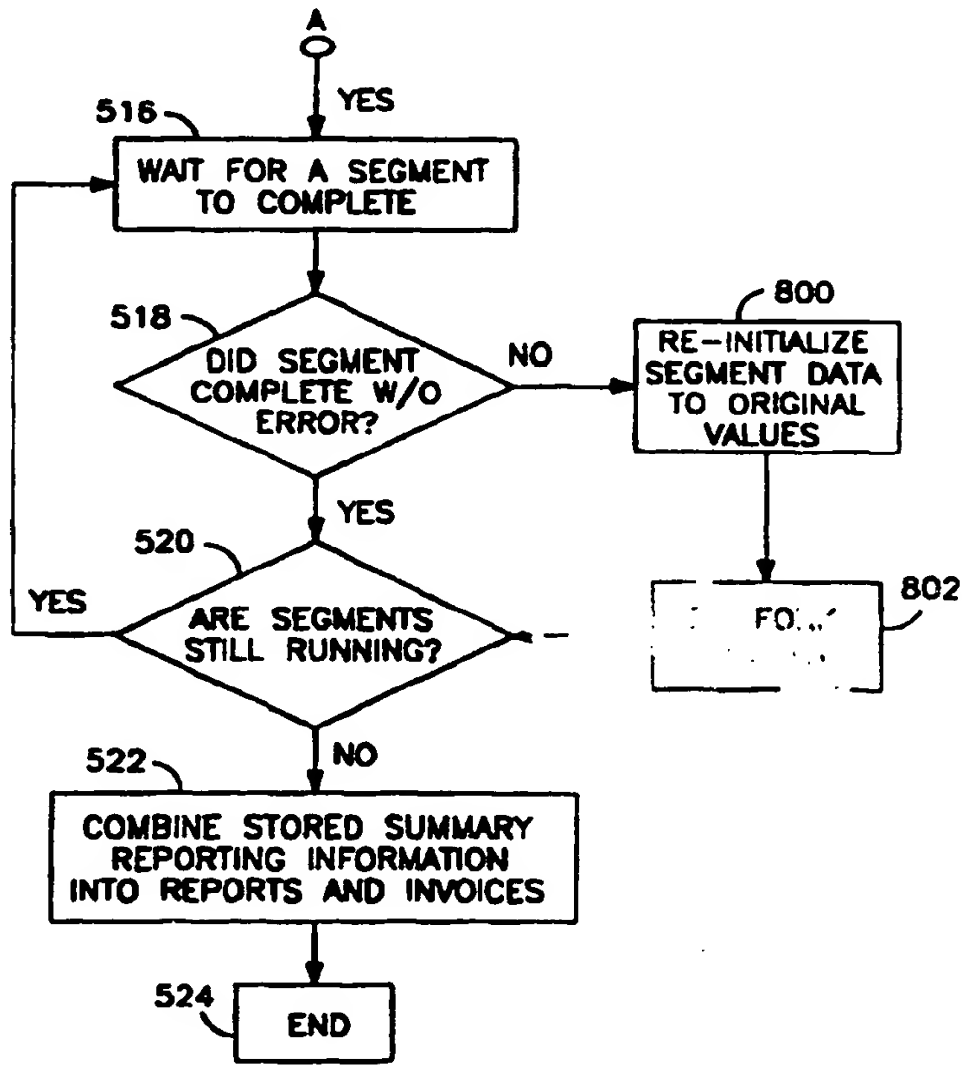
With international search report.

Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: NO-RESET OPTION IN A BATCH BILLING SYSTEM

(57) Abstract

A system and method for processing a batch which is distributed into a plurality of independent segments. A preferred embodiment of this invention calls for implementation on a symmetrical multiprocessing platform, however, the invention is also applicable to massively parallel architectures as well as uniprocessor environments. The batch is broken into segments, wherein each segment comprises a plurality of discrete events, each discrete event comprising a plurality of sub-events to be processed (steps 502-510). The system operates to process each discrete event within each segment sequentially and each sub-event within each discrete event sequentially. The system and method provide for the reprocessing of a failed segment without requiring reprocessing of the entire batch (steps 518, 800, 802).



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	**GB**	United Kingdom	**MR**	Mauritania
AU	Australia	**GE**	Georgia	**MW**	Malawi
BB	Barbados	**GN**	Guinea	**NE**	Niger
BE	Belgium	**GR**	Greece	**NL**	Netherlands
BF	Burkina Faso	**HU**	Hungary	**NO**	Norway
BG	Bulgaria	**IE**	Ireland	**NZ**	New Zealand
BJ	Benin	**IT**	Italy	**PL**	Poland
BR	Brazil	**JP**	Japan	**PT**	Portugal
BY	Belarus	**KE**	Kenya	**RO**	Ro · a. ia
CA	Canada	**KG**	Kyrgystan	**RU**	Russian Federation
CF	Central African Republic	**KP**	Democratic People's Republic of Korea	**SD**	Sudan
CG	Congo			**SE**	Sweden
CH	Switzerland	**KR**	Republic of Korea	**SI**	Slovenia
CI	Côte d'Ivoire	**KZ**	Kazakhstan	**SK**	Slovakia
CM	Cameroon	**LI**	Liechtenstein	**SN**	Senegal
CN	China	**LK**	Sri Lanka	**TD**	Chad
CS	Czechoslovakia	**LU**	Luxembourg	**TG**	Togo
CZ	Czech Republic	**LV**	Latvia	**TJ**	Tajikistan
DE	Germany	**MC**	Monaco	**TT**	Trinidad and Tobago
DK	Denmark	**MD**	Republic of Moldova	**UA**	Ukraine
ES	Spain	**MG**	Madagascar	**US**	United States of America
FI	Finland	**ML**	Mali	**UZ**	Uzbekistan
FR	France	**MN**	Mongolia	**VN**	Viet Nam
GA	Gabon				

BNSDOCID: <WO 9523373A1 I >

NO-RESET OPTION IN A BATCH BILLING SYSTEM

FIELD OF THE INVENTION

This invention relates generally to batch processing and more particularly to batch processing on a symmetrical multiprocessing (SMP) system. This invention has particular application to the batch processing of customer account information in order to perform periodic customer billing.

BACKGROUND OF THE INVENTION

There currently exist systems for customer billing in industries wherein a large number of customers are billed periodically based upon monthly (recurring) charges as well as dynamic use related (non-recurring) charges. Of interest in the design of such systems is the flexibility with which such systems can adjust to changes in such variables as billing structure, tax rates, bill formatting and incentive program implementation. Also of great importance in these systems is the ability to service an increasing number of customers as time progresses.

In a typical billing systems, the system should be designed to interface with peripheral devices and applications providing customer usage data from a variety of sources. In addition, such systems usually allow an employee of the billing company to interact with the system to, for example, specify the time, format and nature of invoice generation.

One example of an industry in which such a billing system is an important part of day to day operations is the cellular telephone/telecommunications industry. In recent years communication via cellular telephones has grown explosively. The requirement for convenient communications has become the norm in business as well as residential markets. Cellular telephones are found everywhere from automobiles and restaurants to golf courses and airplanes. In meeting the challenge of providing quality cellular services to this ever expanding subscriber base, the cellular telecommunications industry has identified a number of issues

which need to be addressed in order to maintain and/or improve customer relations.

A primary concern for a cellular carrier is its ability to provide accurate and easily understood billing statements so that customers will respond promptly with payment and so that customer service interactions may be minimized. In order to achieve this objective, it is often desirable for a cellular carrier to implement such a billing system as a high-volume application with the ability to communicate with applications providing for customer service data entry and retrieval as well as automated data collection mechanisms such as a switch for monitoring customer calls, airtime and toll information. In addition, the overall system may provide fraud detection capabilities, security controls, sales and marketing support, funds collection support and message processing.

Customer service data and applications are generally provided on-line to customer service agents. Typically, the bill summary, bill detail, current balance, payment and adjustment information are available on-line. An agent can view customer information by querying on virtually any field in the database. Customer account information can be altered through customer update screens.

Fraud in areas such as subscription, airtime and roaming fraud have cost the cellular industry millions of dollars over the course of just a few years. In response to this problem a number of security controls have recently been developed for use by the industry. Such security controls include electronic switching networks (ESN's), identification by social security number, mobile number detection and monitoring reports which summarize long distance charges billed versus those recorded at the switch.

With respect to sales and marketing support, the system may provide the ability for airtime, product and other rating promotions to be

BNSDOCID <WO 9523373A1 I >

created through the construction of a new rate plan in the appropriate tables. Access, service and equipment charges, like the rate plans are table-driven. Equipment charges can be categorized as recurring (those that will bill each month for a specified period of time), or non-recurring (one time charges).

Because of the periodic nature of the billing process in the cellular telephone industry, most systems have performed customer billing and invoicing as a sequential batch process. The traditional thinking on how to run the batch process has been influenced primarily by the strengths and weaknesses of the large engine uniprocessor mainframe environment. Thus, batch processes are performed in a "task oriented manner". In other words, each of the component tasks for all of the customer accounts is performed in sequence, prior to the processing of any other component tasks for each of the customer accounts.

Typically, the above-described batch processing has been performed on large scale uniprocessors, such as IBM or DEC brand mainframes which are capable of high throughput. Uniprocessor machines may be provided which operate at about 100 million instructions per second (MIPS). One example of a uniprocessor architecture, although not necessarily operating at 100 MIPS, is the HP 9000 Series 800 Server Family manufactured by the Hewlett Packard Corporation. Fig. 1 depicts the architecture of this machine. As can be seen in Fig. 1, only a single CPU 100 is provided. CPU 100 interfaces, through memory and I/O controller 110, to an expandable RAM storage area 120. A single copy of the operating system will generally reside in main memory 110. System bus 130 is further provided to allow for integration into a local area network or LAN as well as to attach various peripherals desired in order to meet system needs.

BNSDOCID: <WO__9523373A1_I_>

As batched applications comprise a plurality of tasks, and uniprocessor architectures are capable of executing only a single task at a time, uniprocessors are often complimented with special multitasking hardware and operating system software (such as UNIX) which allow the single processing resource to be efficiently distributed among a set of simultaneously initiated tasks. Although this multitasking increases a uniprocessor machine's overall throughput and workflow capabilities, the simultaneously initiated tasks are still in contention for a single processing resource and the amount of execution time allotted to each individual task decreases in proportion to the number of tasks initiated.

To overcome this problem with multitasking, multiprocessor systems, which utilize more than one CPU, have been developed to provide tasks with the same resources offered by their uniprocessor counterparts but further allow these resources to be shared among a set of concurrently executing tasks. In multitasking, multiprocessor environments, various tasks are distributed to the various processors. A fine grain approach parallelizes groupings of similar tasks with all of the tasks being assembled into a finished batch after parallel processing completes. Coarse grain, on the other hand, simply parallelizes groupings of various tasks of the job without regard for the similarity of the tasks within each grouping.

Several multiprocessor systems have become widely used in recent years. Some examples include massively parallel processing systems comprising a plurality of individual processors, each having its own CPU and memory, organized in a loosely coupled environment, or a distributed processing system operating in a loosely coupled environment, for example, over a local area network.

One multiprocessing technology, termed symmetrical multiprocessing (SMP), is a relatively recent architecture that provides applications with a set multiple of CPUs which operate in a tightly-coupled

BNSDOCID: <WO_ _9523373A1 I >

shared memory environment. Many major hardware vendors, e.g., IBM, DEC, HP, NCR, Sequent, Tandem, and Stratus, have released or announced computers that provide this type of architecture and associated processing. SMP techniques and functions have also been provided in some operating systems, such as, for example, an operating system sold under the trademark (MICROSOFT NT) and various derivatives of the multitasking operating system products sold under the trademark (UNIX). In addition, certain databases, particularly relational database management systems, sold under the trademark (ORACLE) and the trademark (INFORMIX) provide features that accommodate SMP techniques and speed up performance in SMP environments.

One significant advantage with an SMP system is scalability. An SMP platform, such as the SMP platforms sold under the trademark(SEQUENT), for example, includes a plurality of tightly coupled individual processors each operating under the control of a shared operating system and each accessing a shared memory. The processors share peripheral devices and communicate with one another using a high speed bus (or special switch), special control lines and the shared memory. A hardware platform designed as an SMP system is generally significantly less expensive than its uniprocessor counterpart for a comparable number of MIPS. This is primarily because of the SMP environments ability to use either a plurality of low cost general purpose CPU's, for example 486-type processors, or mass marketed proprietary processors such as some RISC chips. By contrast, most processors operating in the uniprocessor environment have been specially designed and produced in small quantities and therefore, their price is relatively high. Mass marketing of proprietary processors having broad applications, however, greatly reduces machine cost. Further, the number of processors employed in an SMP environment is variable so that processors can be added to the system as desired. Thus,

BNSDOCID: <WO___9523373A1_I_>

for example, one SMP platform may have 4 processors and another may have 20.

Sequent Computer Systems, Inc. provides one model, the S2000/450 of its SMP platforms sold under the trademark (SYMMETRY), which may include from 2 to 10 processors (typically Intel 486/50Mhz CPUs) and another model, the S2000/750, which may include from 2 to 30 processors. Both models provide from 16 to 512 Mbytes of physical memory with 256 Mbytes of virtual address space per processor. Each processor runs a single, shared copy of Sequent's enhanced version of UNIX which is sold under the trademark(DYNIX/ptx). Specifically, the version 2.0 operating system distributed under the trademark (DYNIX/ptx) is preferred.

For purposes of illustration, a block diagram of the relevant portions of the S2000/750 is shown in Fig. 2. As will be discussed below, a preferred embodiment of this invention is resident on the S2000/750 SMP system manufactured by Sequent Computer Systems, Inc. A system bus 260 is provided to support a multiprocessing environment. System bus 260 is configured as a 64 bit bus and carries data among the systems CPUs, memory subsystems and peripheral subsystems. It further supports pipelined I/O and memory operations. The bus can achieve an actual data transfer rate of 53.3 Mbytes per second. Each processor board 210 in the S2000 system contains two fully independent 486 microprocessors of the type sold, inter alia, by the Intel Corporation. These processor boards (of which there may be up to 15 in the S2000/750) are identical. The memory subsystem 220 consists of one or more memory controllers, each of which can be accompanied by a memory expansion board 270. The controllers are available with either 16 or 64 Mbytes of memory and the expansion boards may add 96 or 192 MBytes.

The Quad Channel I/O Controller (QCIC) 230 board supports up to 24 disks 240, six each on four independent channels. Multiple QCIC boards can support up to 260 GBytes of storage per system. System I/O performance growth can increase as disks are added. A VMEbus interface provides a link to Ethernet 246 and terminal line controller 248, among other possibilities. Further, the ability to add a parallel printer 252 is provided through system services module 254. Finally, a SCSI interface is provided for integration with various secondary storage devices 258.

In the past, symmetrical multiprocessing platforms such as those manufactured by Sequent have been utilized primarily for processing individual events, such as in an On-Line Transaction Processing (OLTP) environment. OLTP systems frequently involve large databases and interaction with a plurality of users, each typically operating a terminal and each using the system to perform some function requiring a predictable response within an acceptable time. In such an environment, interactive user requests, such as may be provided in customer service systems, are processed. Because each of the user requests is typically independent, the SMP system is particularly effective.

When processing batches, the processor generally performs as many similar operations as possible in one job step in order to achieve high throughput for a logical batch run. For example, in a bill processing environment, if bill processing was oversimplified to comprise four tasks, for example,:

1) process payments to account;

2) process charges to account;

3) calculate taxes for account based upon charges; and

4) print invoices to disk.

BNSDOCID: <WO__9523373A1_I_>

in the traditional batch environment of the prior art, the batch job steps would be as follows:

1) initialize and start job;

2) sort payments by account number;

3) sort charges by account number;

4) process all payments for each account in account number order;

5) process all charges for each account in account number order;

6) process all tax calculations for each account in account number order;

7) print all invoices to disk; and

8) end job.

In this environment, if one of the individual processes fails, then the entire batch must be re-processed. Failures occur in even the most effective systems.

A need has arisen for a system which can determine and re-execute groupings of discrete events within a failed batch without having to re-process every discrete event in the batch. Additionally, a need has arisen for a scalable computer architecture capable of effectively and efficiently processing customer account billing statements.

SUMMARY OF THE PRESENT INVENTION

In view of the foregoing, it is a principle object of this invention to provide a system and method for efficiently executing batch runs.

It is another object of this invention to provide a system and method for efficiently executing batch runs in an SMP environment.

It is an object of the present invention to provide a system for running batch jobs efficiently even if failures occur.

It is a further object of the present invention to re-process only improperly processed portion of the batch after a failure.

The present invention subdivides each batch process into segments. The segments execute in a multi-tasking environment as separate processes, yet integrate, upon conclusion, as a single batch entry for continuing processing. The present invention provides significantly improved resource utilization especially when multi-channel access to the memory storing the discrete events is provided.

The present invention provides the advantage of being portable to any or multiples of the available multi-tasking hardware architectures and configurations, from low cost personal computers running the multitasking operating system sold under the trademark (UNIX) to tightly coupled SMP architectures, to loosely coupled massively parallel architectures, all of which may implement at least linear performance scalability as needed by adding I/O channels, machines and/or processors.

Further, the present invention provides a system for exploiting the capabilities of a symmetrical multiprocessing system in a batching environment. Implementing the segments of the batch on an SMP platform provides logarithmic scalability in an individual cabinet. Further, as additional cabinets are added, logarithmic scalability may still be attained. However, as cabinets are added, the point at which the performance increase gained through additional processors tapers off will occur at a higher number of processors. For example, in a system operating with only one cabinet, the number of transactions processed increases linearly as additional processors are added up to X number of processors, for example 20, at which point adding processors increases throughput less than linearly. By adding a second cabinet, linearly performance increase is achieved for X + Y number of processors, for example 38. In addition, because of the

BNSDOCID: <WO___9523373A1_I_>

scalability of the SMP architecture, optional growth paths based on performance/capacity requirements and budget limitations allow for efficient processing of batch jobs. Very high performance is therefore provided at a relatively low cost.

The present invention also provides a system which enables a batch to be distributed into a plurality of independent segments. Each segment comprises a plurality of discrete events, each discrete event comprising a plurality of sub-events to be processed. The system operates to process each discrete event within each segment sequentially and each sub-event within each discrete event sequentially. The plurality of segments may be processed on an uniprocessor, an SMP system, a massively parallel processing system or a distributed loosely coupled system. By balancing the number of discrete events in each segment using a data segment parallelized "coarse grain" approach, a flexible but efficient use of processor availability is obtained.

BRIEF DESCRIPTION OF THE FIGURES

Fig. 1 is a system block diagram of a conventional uniprocessor platform according to the prior art.

Fig. 2 is a system block diagram of an SMP platform according to the prior art.

Fig. 3 depicts a block diagram of the overall system organization according to a preferred embodiment of the present invention.

Fig. 4 is a flowchart of an overall computer system operation according to a preferred embodiment of the present invention.

Fig. 5A-5B is a flowchart of a process of multi-threaded batch processing according to a preferred embodiment of the present invention.

Fig. 6A-6B is a flowchart of a process for distributing discrete events into a plurality of segments according to a preferred embodiment of the present invention.

Fig. 7 is an example sequence of a plurality of discrete events in a batch to be processed according to a preferred embodiment of the present invention.

Fig. 8 is a flowchart representing a no-reset process for processing discrete events according to one preferred embodiment of the present invention.

Fig. 9 is a flowchart representing a no-reset process for processing discrete events according to another preferred embodiment of the present invention.

Fig. 10 depicts a block diagram of a loosely coupled distributed processing architecture according to another preferred embodiment of the present invention.

Fig. 11 depicts a block diagram of a massively parallel processing architecture according to yet another preferred embodiment of the present invention.

Fig. 12 depicts a block diagram of a linked computing system utilizing a massively parallel processing system, symmetrical processing system, uniprocessor system, and loosely coupled distributed processing system according to another preferred embodiment of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Fig. 3 depicts generally a computer system according to a preferred embodiment of the present invention. For exemplary purposes, a system for use in the cellular phone industry will be described without limiting the scope of this invention. The computer system comprises a front end processor 300 and a back end processor 310. Front end processor 300

and back end processor 310 communicate via communication link 314. Front end processor 300 is also connected to a switch 306, which has a switch memory 316 connected thereto, at least one customer service node 308 and a disk farm 302. Back end processor 310 is connected to a printer 318 and a disk farm 304.

Front end processor 300 communicates with at least one switch 306 to activate or deactivate customers. Switch 306 may be any type of switch commonly used in the cellular phone industry for determining when calls are placed and for making call connections. In order to determine which customers places a call, a listing of active customers for that particular switch is maintained at each switch 306. When a customer attempts to place a call in his or her home market, that customer's ID is compared with the active customer list for the home market. If the customer's ID is not in the list, then the call is not processed. Verification of customer calls placed from outside the customer's home market, a process called roaming, occurs through another process. In order to maintain the customer list accurately, front end processor 300 notifies switch 306 via switch interface software 322 operating on front end processor 300 whenever a customer is activated, deactivated, suspended or re-stored or whenever a customer's custom calling feature is added or changed or the customer's service number is changed. Switch interface software 322 is responsible for updating and verifying the customer lists at switch 306. Periodically switch interface software 322 runs comparisons between the customer lists at switch 306 and that at front end processor 300 to verify correspondence between the two. Front end processor 300 also communicates with at least one customer service node 308. These customer service operations are commonly referred to as On-Line Transaction Processing or OLTP operations 324 which are performed in response to commands or requests from customer service nodes 308. OLTP

operations 324 in general are known, and in the cellular phone industry comprise a variety of customer service functions as will be described in detail below. Back end processor 310 connects to printer 318 to generate hard copy invoices to be sent to customers.

Front end processor 300 and back end processor 310 communicate via communication link 314. This interface may be, for example, an SQL*NET connection (TCP/IP). Alternatively, front end processor 300 and back end processor 310 may comprise one processor and may communicate through a database transfer between databases maintained by each processor.

Two major databases are used by the customer billing system. These databases typically store large volumes of information and therefore are preferably relational database management systems. The RDBMS sold under the trademark(ORACLE7), is preferable for managing the data in these two databases. A first database, a customer information database 312, is stored in disk farm 302 at front end processor 300. Customer information database 312 stores all of the information regarding the customer including accounts receivable, charges, service plans, etc. The second database, an invoice processing database 320, resides in disk farm 304 associated with back end processor 310. Invoice processing database 320 stores all of the information necessary for invoice processing, including customer information and customer calls which are downloaded from front end processor 300 and switch memory 316 respectively, as described in detail below.

These databases may reside on one disk, or may be spread out over several disks in disk farm 302 and 304 respectively. One preferred method for spreading the database over a plurality of disks is a method called striping. Striping is a method whereby pieces of information from a particular entry into the database are spread onto several disks. This

BNSDOCID: <WO___9523373A1_I_>

process is especially useful so that memory accesses will occur to a plurality of disks instead of the same disk all of the time. This provides for reduced I/O time because the same I/O channel is not being constantly barraged with requests; rather, those requests are spread over a number of I/O channels.

The operation and interrelation of these components will be described with reference to Fig. 4. Fig. 4 is a flow diagram representing an overall system function according to a preferred embodiment of the present invention. Two basic operations occur simultaneously in two separate locations, at switch 306 and at customer service nodes 308. In step 400, cellular calls are received by switch 306. Because customers place cellular calls at all times of day and night, this is a continuous process. As these calls are received, switch 306 generates customer call information for storage in step 402 into switch memory 316. As indicated in Fig. 4, this process occurs entirely at switch 306.

Periodically, as shown at step 404, customer call information is transferred from switch 306 to back end processor 310. Switch 306 may be located in a remote location with respect to back end processor 310 and therefore, the data collected and stored onto switch memory 316 is transferred to back end processor 310 via any communication link or stored onto a tape or other form of secondary storage and manually transported to the location of back end processor 310 for uploading. For example, switch 316 may offload the customer call data onto a tape and the tape may be mailed, or sent via an express courier service to back end processor 310 and then downloaded onto disk farm 304. Alternatively, a data link could be established between switch 306 and back end processor 310 for transferring this information over a link.

In one preferred embodiment, one front end processor 300 may service a plurality of switches 306. Therefore, rather than having a

plurality of data links, manual transfer may be preferred. Step 404 occurs periodically. Preferably, customer call information is transferred daily, although weekly or even monthly transfer is possible. Daily transfer is preferable because the data which is generated over an entire month may be extremely large and transferring, as well as processing, that volume of data would likely take an unreasonable amount of time.

After the customer call information is stored at back end processor 310, each call is rated in step 406. Rating, or message processing, is a function to format each customer call and then associate certain charges associated with each customer call. In step 406, messages, or customer calls, from a variety of cellular switches may be processed and placed in a standard format. Messages are edited to determine conformity with a standard format and those which do not pass basic edits are placed into files for review and re-processing by a user at back end processor 310. A control report may be generated from the messages which cannot be processed further and those messages may then be edited and returned for processing.

Message processing also involves generating message statistics. With each set of messages uploaded at back end processor 310, summary statistics may be compiled regarding completed calls, incomplete calls, free calls, toll calls, local calls, special calls, records in error, and total records processed. Records corresponding to switch interconnections are also counted in these report records and are reported separately as well.

Each message is compared to a npa/nxx-based local coverage area. Each local coverage area is defined separately based on the customer's home region, foreign carrier region, single cell site, group of cell sites and/or service plan, each of which may be changed from time to time through OLTP operations 324 operating through customer service node 308. The coverage area is composed of groups of npa's, npa/nxx's, or even

npa/nxx/lines. These groups can be combined into "super groups" to form coverage areas which can be easily manipulated to support a wide variety of local calling area scenarios.

After each call has been compared to a local coverage area to determine the origination of the call, toll rating is performed. Toll rating is based on the origination npa/nxx, destination npa/nxx, and time and date of the call (peak, off-peak, weekday, weekend, holiday). Toll rating is preferably based on tables supplied and updated by a third party supplier, for example, Q-TEL 9000, and can be modified on an as-needed basis. The toll rating system from Q-TEL provides toll rating information for all calls originating from all United States toll carriers. Another toll rating system is required to process calls placed outside the United States.

Rating also involves toll discounting. At any time, toll charges may be discounted based on the cell destination and the specific day or date rate. Day or date range discounting can be easily accomplished through menu options and is based on customer region or service plan. For example, toll charges might be discounted 50% for January 1993 for all customers on a particular service plan in order to support a special marketing promotion. Call destination discounting is accomplished by creating a special toll coverage area. Toll discounts may be assigned based on the call destination npa, npa/nxx or npa/nxx/line. Further, these toll discounts can be combined with local coverage areas to form sophisticated local calling area and toll discounting scenarios to support campus or PCN configurations. Thus, rating is performed on each call to determine a set rate to charge for that particular call and any flat fees to charge to each call. Once rating is performed, the rated calls are stored in step 408 to disk farm 304 in flat file format. As new groups of calls are rated, the rated calls are simply appended to the end of the flat file.

In step 410, after each group of rated calls is stored, back end processor 310 checks to see if the user has requested that customer invoices be generated. If not, then the loop of receiving calls in step 400, storing the calls in step 402, transferring the calls to back end processor 310 in step 404, rating calls in step 206 and storing the rated calls in a flat file in step 408 continues. Alternatively, if the user has requested that the invoice generation begin, then in step 412, the rated calls stored in the flat file are transferred into invoice processing database 320 and a new flat file is used to store the rated calls from step 408. The rated calls stored in invoice processing database 320 are then ready for invoice processing as performed in step 422 by the multi-threaded batch processor as will be described in detail below. The information stored into invoice processing database 320 may comprise, by way of illustration only, the information depicted in Table 1 below.

FIELD NAME	FIELD TYPE
Customer Account Number	CHAR(10)
Area Code Called	CHAR(3)
Exchange Called	CHAR(3)
Line Number Called	CHAR(4)
System ID Where Call Originated	CHAR(5)
Date of Call	DATE
Start Time of Call	CHAR(6)
Duration of Call	NUMBER(10)
Time of Day Toll Rating	CHAR(1)

BNSDOCID: <WO___9523373A1_I_>

Time of Day Airtime Rating	CHAR(1)
Call Type - Mobile to mobile, Land to Land, Mobile to land, Land to Mobile, etc.	CHAR(1)
Taxes	NUMBER(10)
Batch Id Number	CHAR(20)

Table 1: Example of Customer Call Information Passed to Invoice Processing Database at Back End Processor 310

Concurrent with the call processing operations of steps 400-412, customer information is periodically received in step 414 at front end processor 300 through customer service nodes 308. For each customer, a record of information is maintained in customer information database 312 stored in disk farm 302 at front end 300 and may include the customer's name, address, telephone number, credit rating, home cell, service plan, customer calling features selected, any custom calling information, accounts receivable, etc. This information is accessible by customer service representatives through customer service nodes 308. As customers request changes or request status inquiries, customer service representatives may enter the changes or requests via OLTP operations 324 running on front end processor 310.

Payments are also entered through OLTP operations 324 in step 414. As bills are paid, the payments received are entered into a customer information database through OLTP operations 324. Payments are posted in real time, automatically updating bill balance information for each customer and for collection queues which exist to track delinquent

customers. Customer information access is also provided through OLTP operations 324. Deposits and other payments may also be recorded into customer information database 312.

Collections information may also be updated regarding each customer in step 414. When an account meets a table-defined collections criteria, for example, based on the length of time the account has been past due and the credit class assigned for the particular customer, that account is selected for collections activity. These accounts are prioritized and placed in a collections queue for a collections agent. Customer service managers may also change the order of or add accounts to the collections queues if desired.

A collections agent located at a customer service node 308 is then responsible for entering a result code to categorize the response received when the customer whose payment is past due has been contacted. This information is also received in step 414. If the result code is a busy or no answer, then the customer account is placed back into the queue. If, for example, a customer promises to pay by a certain date, that date is entered by the collector and received by the front end at step 414. Then, if payment is not received by a specified number of days past that date, the account automatically returns to the collection queue for resolution. Comments may also be stored for each customer account and stored into customer information database 312 through steps 414 and 416.

Other information may also be entered in step 414, such as tax information, one time charges, paybacks, adjustments, etc. All of the types of information concerning the customer may be changed through the customer service nodes 308 and front end processor 300 operating under OLTP operations 324.

As customer information is received in step 414, front end processor 300 stores this information into customer information database

BNSDOCID: <WO___9523373A1_I_>

312 in step 416. In step 418, front end processor 300 checks to see if a system operator has selected a "cut-off" in customer service information. A "cut-off" is selected by a system operator, for example, a supervisor for the cellular phone billing company who wishes to complete the charges for a particular billing period. Access to cut-off is preferably limited to a selected few individuals at each of the locations to avoid generating invoices errantly. If cut-off has not been selected, the loop of receiving and storing customer information continues. Alternatively, if cut-off has been selected in step 418, then in step 420, all customer information necessary for billing is dumped from customer information database 312 at front end processor 300 to invoice processing database 320 at back end processor 310. The customer information which is dumped generally comprises information, for example, as depicted in Table 1 below. After the customer information has been dumped into invoice processing database 320 at back end processor 310 across communications link 314, invoice processing may take place in step 422 under the control of the multi-threaded batch processor.

FIELD NAME	FIELD TYPE
Customer's Last Name	CHAR(15)
Customer's First Name	CHAR(15)
Customer's Social Security	NUMBER(9)
Account Type - Individual,	CHAR(1)
Customer Status - Pending,	CHAR(1)
Rate Plan	CHAR(3)
Credit Class	CHAR(1)
Service Plan	CHAR(1)

BNSDOCID: <WO___9523373A1_I_>

Accounts Receivable	Data structure
Adjustments	Data structure
Custom Calling Features	Data structure
Recurring Charges	Data structure
Non-recurring charges	Data structure
Refunds	Data structure
Deposit	Data structure

Table 2: Example of data passed from Customer Information Database at Front End Processor 300 to Invoice Processing Database at Back End Processor 310

As depicted in Table 2 above, a large volume of data is collected and stored regarding each customer. A database management system which is particularly effective at handling this large amount of information is important. Relational database management systems are particularly useful. In one preferred embodiment, customer information database 312 is a relational database management system, for example the relational database management system sold under the trademark(ORACLE7) which provides simple operations which perform the mass transfer of database information from one database to another. By simply executing these operations the transfer of data may be accomplished. As discussed below, a relational database management system is preferable additionally because of its ability to support SMP functions. Relational database management systems such as those sold under the

BNSDOCID: <WO___9523373A1_I_>

trademark(ORACLE7) are particularly suited to SMP functions on a Sequent platform.

Fig. 5A-5B depicts a flow diagram of the operation of the multi-threaded batch processor (MTBP) according to a preferred embodiment of the present invention. The steps discussed with respect to Fig. 5 correspond to the overall process step 422 depicted in Fig. 4. One embodiment, implemented in source code, of a multi-threaded batch processor according to the preferred embodiment is contained in pages 2-31 of the accompanying Microfiche Appendix.

Batch processing, as described above, comprises processing a plurality of discrete events. As used herein, the term "discrete event" is not limited to a time specific occurrence, but rather a collection of data which is to be processed to generate an outcome. Preferably, each discrete event comprises a plurality of sub-events. Each sub-event is also data which is to be processed. In a preferred embodiment of the present invention, when applied to the cellular phone customer account processing, each customer account to be processed is treated as a discrete event. Further, since a number of details regarding the customer account must be processed, for example, recurring charges, non-recurring charges, taxes, customer calls, accounts receivable, etc., each of the details is treated as a sub-event which may be processed separately.

Upon start up in step 500, MTBP requests input of the batch by requesting the batch identification, the number of segments into which the batch is to be distributed and the requesting user's identification and password. If any of these parameters is invalid, then the batch is aborted in step 528. According to an alternative embodiment, instead of the number of segments being requested and input by a user, this information could be automatically determined by the MTBP.

According to this alternative embodiment, the number of CPU's operating on the system may be determined through a system call. Therefore, if the system were operating in a uniprocessor environment, then the number of CPU's would be equal to one. If the system were operating in a SMP or MPP environment, then the operating system could pass this information as well as the number of active processors to the MTBP. In one preferred embodiment, the number of segments then could be selected to be equal to the number of CPU's. Other algorithms for selecting the number of segments as compared to CPU's are also within the scope of the present invention. For example, the number of segments could be selected to be some number times the number of CPU's operating. There could also be eight (8) CPU's and 10 segments, for example. The number of segments need not correspond to the number of CPU's

Once the number of segments has been input either by a user or determined automatically and all parameters have been determined to be valid, in step 504 the account number range is determined from the batch identification. The account number range then is checked to ensure that valid accounts exist within that particular range. This step is performed as a "double-check" to validate the account range specified. Errors that sometimes might occur are specifying an account range outside of the range of existing customers or specifying a range wherein certain events are selected to be within the range but other dependent accounts are not. If all of the parameters are valid, as checked in step 504, then control passes to step 506 to determine if valid accounts exist within that particular account number range. If so, then the job is aborted in step 528. If valid accounts do exist, however, then in step 508 the MTBP determines the number of accounts per segment.

Step 508 of loading the segments may be accomplished in multiple ways. It may be desired to load each segment with similar types

BNSDOCID: <WO___9523373A1_I_>

of customers. Alternatively, it may be desired to load each segment with a wide range of customers. Another possibility might be to load more active customers into a segment which contains fewer customers to be processed. One preferred embodiment of the MTBP for distributing the accounts into the various segments is represented by the flow diagram of Fig. 6 which will be described in detail below.

After each segment has been loaded with a set of customer accounts to be processed, a first segment is forked in step 510. "Fork", as will be recognized by one of ordinary skill in the art, is a function based on the multitasking operating system sold under the trademark(UNIX) representing the activation of a child process on the processor. While the MTBP acts as a parent process, each fork call generates a child process corresponding to a segment. When a segment is forked, resources are allocated to that segment and the segment begins sharing processor time with other active processes. "Fork" is a system call generic to the environment of the multitasking operating system sold under the trademark(UNIX) which creates a new process having a duplicate copy of the parent process, here the MTBP, with the same data, register contents, and program contents as the parent. The forked process, here the segment, has access to the same files as the MTBP. Also in step 510, the "exec" system call is made. "Exec" starts the newly created process generated by the "fork" system call. If the first segment is forked and started successfully, as detected in step 512, then the MTBP determines in step 514 if all of the segments have been forked. If the forking and starting in step 510 was unsuccessful, then the batch is aborted in step 528.

After all of the segments have been forked, then the MTBP waits in step 516 for a segment to complete processing. If a segment completes processing, the MTBP verifies whether the process completed without error and if so continues to step 520. If an error was detected, then

BNSDOCID: <WO __ 9523373A1 I_>

the remaining segments are terminated in step 526 and the batch is aborted in step 528. In step 520, the MTBP checks to see if some of the segments are still being processed. If so, then it continues to loop though steps 516, 518 and 520 until all of the segments complete.

If all of the segments complete without error, then in step 522, the entire batch of customer accounts may be combined, summary reporting may occur and invoices may be generated for each of the customer accounts. Often an entire batch must be approved before invoices are mailed. Thus, by collecting each of the segments back into the batch, summary approval may be performed. After summary processing is complete, the MTBP terminates in step 524 until the next batch of customer accounts needs processing. Summary processing may include generating a detailed summary or bill detail for use by the system operator in deciding whether to approve the bill or not. Also, invoice generation performed as a step performed in step 522.

One method of placing the customer accounts into the various segments which are generated is depicted in Fig. 6. As shown in Fig. 6, in steps 610 through 616 the number of segments is determined. This may alternatively be input by a user. Once the number of segments to be used has been determined, then the number of customer accounts to be distributed into those segments must also be determined in step 618.

In general, customer accounts are labeled as either an individual account, a master account or a child account. This type of labelling is necessary in order to differentiate between individual customers and aggregate accounts such as company sponsored customers for whom cumulative company bills and individualized child bills must be generated so the company can monitor use by each of its employees.

Therefore, companies are typically labeled as master accounts and receive an aggregate bill for each of their employees. The employees

are labeled as child accounts. In order to process a master account, therefore, each of its child accounts must be processed first. In order to signify the beginning of a master and child grouping, master customer ID's are placed sequentially before child customer ID's in the account number listings which are produced for batching. When distributing customer accounts into the various segments, it is desirable to keep the child accounts and the master accounts in the same segment. Once the number of customer accounts to be processed and the number of segments have been determined, then the number of customer accounts per segment is determined in step 620 to simply be (number of customer accounts) DIV (number of segments). Further, in step 622, a remainder is determined to be equal to (number of customer accounts) MOD (number of segments). After the remainder has been determined, in step 624 a number of segments equal to the remainder are selected and in step 626, the segment size for each of those segments is incremented by one. Thus, for example, if there were 7 segments and the remainder was 4, then the first four segments may be selected and the segment size of each may be incremented by one.

After each segment has a segment size assigned to it, in step 628 the segments are ordered from a first segment to a last segment for distribution of the customer accounts. In step 630 the first segment is selected as the current segment and in step 632 a number of customer accounts, or discrete events, equal to the segment size for that particular segment are distributed sequentially from the account number listing provided in the batch into the current segment. At this point, in step 634 it is determined whether the last customer account which was distributed into the current segment was a child account. If not, then control passes to step 642 to determine if all of the segments are full. If the last distributed customer account was a child account, it is determined whether the next

BNSDOCID: <WO___9523373A1_I_>

customer account to be distributed is also a child account. If not, then control passes to step 642. If it is, then in order to process the child and master account in the same segment, the next child account is added to the current segment. This step is performed in step 638. Since adding another customer account increases the size of the current segment, the size of one of the other segments must be decremented so that the total distribution of customer accounts will correspond to the sum of the segment sizes for all of the segments. Thus in step 640, the next segment's segment size is decremented by one. It is possible that the next segment's segment size could be reduced to zero or even below zero. This would occur for a very long distribution of a child account for one particular master account.

If this were to occur, then a number of schemes to correct this problem are possible. For example, one less segment could be used. Alternatively, other load balancing schemes could be employed. However, since each batch may typically include about 10,000 or more customers and often anywhere between 2 and 20 segments are used, the likelihood of getting a master and child grouping which is over 5000 or even 500 is unlikely. Once the next segment's segment size has been decremented by one in step 640, control passes back to step 636 to continue to check to see if all of the child accounts in the sequential listing of child accounts has been added to the current segment.

In step 642, it is determined whether the current segment is the last segment to have been distributed customer accounts. If it is, then the process ends in step 646. If not, then in step 644, the next segment is selected as the current segment and the loop through steps 632, 634, 636, 638, 640 and 642 continues until all of the segments have had customer accounts distributed to them.

To further illustrate this process, reference will be made to an example customer account grouping in Fig. 7. Fig. 7 depicts an example

BNSDOCID: <WO__9523373A1_I_>

of an account number listing. I indicates an individual customer, M, a master customer and C, a child customer. If 3 CPU's were available and no user override were active, then in step 614, the number of segments would be set to equal 3. Therefore, the MTBP would create three segments, for example, segment1, segment2, and segment3. In step 618, it would be determined that there were 13 customer accounts to be processed in the listing in Fig. 7. In step 620, the number of customer accounts per segment would equal 4 and in step 622, the remainder would equal 1. In step 624, segment1 would be selected and its segment size, which was equal to 4, would be incremented to 5. Thus, we would have segment1 with a size of 5, segment2 with a size of 4 and segment3 with a size of 4.

In step 628, the segments would be ordered segment1, segment2, segment3. In step 630, segment1 would be selected as the current segment. Since segment1 has a segment size of 5, then the first 5 customer accounts (customer accounts 1-5) would be distributed into segment1. In step 634, it would be determined that the last customer account (customer account 5) distributed is a child account. Then in step 636, it would be determined that the next customer account to be distributed is also a child account. In step 638 then, customer account 6 would be added to segment1 and in step 640, the segment size of the next segment, segment2 will be decremented from 4 to 3.

Returning to step 636, once again it would be determined that the next customer account to be distributed (customer account 7) is a child account. Again, customer account 7 is added to segment1 and segment2's segment size is decremented from 3 to 2. After progressing through the distribution in this example, segment1 would have customer accounts 1-7, segment2 would have customer accounts 8-11 and segment3 would have customer accounts 12-13.

BNSDOCID: <WO__.9523373A1.I_>

As this example illustrates, this particular scheme may not balance the load particularly evenly. Other balancing schemes could therefore be used based upon the expected configurations of data in order to finely balance the number of customer accounts in each segment.

In a preferred embodiment, both front end processor 300 and back end processor 310 comprise SMP systems. Because the present invention processes each customer account as a discrete event, the advantages of the SMP architecture may be exploited. Particularly, because each customer account is treated as a discrete event, each discrete event can be processed independent of the other discrete events and thus utilize as many CPU's as are available. The throughput for processing discrete events according to the present invention may vary when running on uniprocessing systems, massively parallel processing systems, loosely coupled distributed processing systems depending, inter alia, upon the number of I/O channels, total CPU speed, and system bus capabilities. One preferred embodiment, implemented in source code, of the present invention's billing operations which operates on either an SMP, uniprocessor, a massively parallel processing system or a distributed loosely coupled system is contained in pages 32-68 of the accompanying Microfiche Appendix.

For example, in the cellular phone environment, for each customer account, there may be three tasks which must be performed in order to generate the bill for the customer account. If those tasks were to 1) process payments, 2) process charges, and 3) process taxes, then for the first customer account, the customer's payments would be processed, then the customer's charges would be processed, and then the customer's taxes would be processed. Once all three tasks for that particular customer are complete, that particular customer can be stored to the processed data memory location, possibly another database and the next customer account in the segment can be processed.

Processing in this manner makes the system model function more like an OLTP model in which there are many events being performed concurrently. This type of model efficiently processes on an SMP machine running with an RDBMS database. Since a uniprocessor can also perform OLTP (albeit less effectively), the discrete customer system works fairly well in that environment as well.

It is possible to run the discrete customer events just like a traditional OLTP system in which each discrete event is a process and each process is placed in the queue waiting for the next available processor. In that instance, one process would be forked for each customer account and the system and database would handle these transactions transparently.

In order to obtain the control necessary to optimally tune the system, however, in a preferred embodiment, these discrete events are distributed into a plurality of segments as discussed. Each segment then is processed as a separate process and is forked off in that manner. Within each segment, each discrete event is processed sequentially, never beginning processing of the next discrete event until the active discrete event has finished processing. Further, since processing for each discrete event generally comprises a plurality of tasks, each task is also sequentially performed until all of the tasks for the active discrete event are complete. Because a smaller number of segments are forked as processes, the user can moderate the amount of resources used by the process to match the current hardware configuration as well as transparently meet future growth.

Existing billing applications which process customer accounts can be easily modified for use in the present distributed processing environment. Batches can be processed at many sites and on many different physical machines and still allow for information on all batches to be reported on or queried via the inherent functionality of RDBMS's.

BNSDOCID: <WO___9523373A1_I_>

One preferred embodiment of a billing system operating as a discrete event processor according to the present invention comprises using a SMP platform sold by Sequent under the trademark(SYMMETRY) model number S2000/750 running the relational database management system sold under the trademark(ORACLE7) and running version 2.0 of the operating system sold under the trademark(DYNIX/ptx). Treating each customer account as a discrete event, software operating according to the present invention in one embodiment generated 50,000 customer invoices in 64 minutes. In this embodiment, twenty four (24) 486 CPU's sold under the trademark(INTEL) operating at 50MHz were provided. Main memory comprised 576 MBytes with each CPU having a cache memory of 512 KBytes. Twenty Four (24) disks in the disk farm comprised 1.3 GBytes of memory each with a disk having 300 MBytes located at the back end processor. In that process over 4,000,000 cellular telephone calls were processed as well as 7,503 adjustments to customer information, 10,004 commission wavers, 47,519 current charges, 32,513 payments and 20,008 non-recurring charges.

The present invention may alternatively operate in a uniprocessor environment, such as an HP 9000 Series 800 Integrated Business Server running the relational database management system sold under the trademark(ORACLE7) and version 9.0 of the operating system sold under the trademark(HP-UX). As discussed, the processing occurs with each segment operating as a separate process, but with each discrete event being processed sequentially within each segment as discussed above.

This invention may similarly be practiced in a loosely coupled distributed computing environment. An example of such an environment is illustrated in Fig. 10. In a distributed computing architecture, a plurality of workstations, PCs or other small computing systems may be connected so that they share data through a wide area network (WAN) or a local area

BNSDOCID: <WO__9523373A1_I_>

network (LAN). In Fig. 10, four computing systems are shown connected via LAN. For purposes of illustration, each computing system 1000 and 1010 could be a personal computer having, for example, a 486-type CPU and one or more hard drives 1040 and 1050. The communication network could be, for example a LAN sold under the trademark (ETHERNET). Preferably, system 1000, having the single copy of the master billing process 1020, should be a higher MIPS machine such as a file server.

As mentioned, one copy of the master billing process 1020 resides on system 1000 and master billing process controls each of the distributed billing segments 1080 resident on systems 1000 and 1010. Each system may further have other applications 1090 resident in main memory and executing concurrently with billing segment 1080. Each system 1010, must have customer data for the appropriate segment moved to local disk 1050 prior to the bill run. Master billing process 1020, containing the multithreaded batch control system, described above, distributes discrete events into segments which are distributed to the local systems 1010. The multithreaded batch processing system when operating on a massively parallel processing system or a loosely coupled distributed processing system would include mechanisms to initiate and distribute tasks among the various processing resources within the system. Each system will then run the segment it has been assigned. Master billing process 1020 will monitor the running systems to determine if any segments have failed and need to be rerun. At the completion of all segments, the master billing process will produce reports and execute additional billing segments from a new batch should it be necessary.

The present invention may also be employed on a massively parallel processing system. Massively parallel systems employ dozens to hundreds or even thousands of processors to provide performance rates rivaling the rates provided on even supercomputers.

BNSDOCID: <WO 9523373A1_I_>

By adding more processors into this loosely coupled arrangement, scalability is provided almost infinitely. In this environment, even tens of thousands of MIPS are possible using, for example, general purpose CPU's such as 486 systems.

Fig. 11 depicts a massively parallel architecture. As depicted, a plurality of CPU's 1100, each having an associated memory 1102 are connected to each of the other CPUs 1100 via connections 1104. By having direct communication with each of the other CPU's, a loosely coupled arrangement is obtained.

One example of a massively parallel machine on which the present invention may operate is the supercomputer sold under the trademark(nCUBE2). This machine employs from 8 to 8192 nodes each consisting of a single chip 64 bit computer and memory chip. The proprietary microprocessor sold under the trademark(nCUBE2)is rated at 7.5 MIPS, so system performance may scale up to over 60,000 MIPS. The system provides over 512 GBytes of online memory with an interconnect communication throughput of 270 GByte/sec. This system may utilize the relational database management system sold under the trademark(ORACLE7). Multiple database servers run concurrently and independently, each of the CPU's 1100 having its own. However, even of the processing servers may process the requests of each of the CPU's 1100. Each parallel server has its own parallel cache manager to track the current location of database blocks in the caches of all of the servers. Most of these functions are provided through the relational database management system and the present invention, therefore takes advantage of the features present in the management system to effectively process the batch job over the massively parallel processing system architecture.

In another embodiment as depicted in Fig. 12, a network 1200 may be connected to the massively parallel architecture depicted in Fig. 11

and shown as 1208. In this embodiment, a plurality of minicomputers 1202, PC's 1204 and/or UNIX workstations 1206 may be connected to the network 1200. Each of the systems 1202, 1204, 1206 could alternatively comprise a loosely coupled distributed processing system or a symmetrical processing system. Networking capabilities provided by the relational database management system sold under the trademark(ORACLE) such as SQL*NET technology utilizing communications protocols such as TCP/IP and DECnet are used to utilize the resources of the massively parallel architecture 1208. In this configuration, the MTBP may reside on either a PC, workstation or one of the CPU's 1100 included in the massively parallel architecture 1208.

Even the most effective systems run into failures while running batch jobs. When a batch terminates abnormally, heretofore, the entire batch was re-processed. In one embodiment according to the present invention, a system is provided for eliminating the need to re-process an entire batch if only a few of the discrete events within the batch fail. In the billing context, this may mean that only one or more customer accounts did not process correctly. The problems involved in this process result from the need to re-process the failed segments or customer accounts and still be able to reassemble the originally successfully processed segments and the re-processed segments into a batch for approval and invoicing.

In order to accomplish this function, it is necessary to maintain a secondary set of tables associated with the customer information and reference data (information contained in customer information database 312). These tables will be used to hold all data created by the billing process, as well as any changes to erroneous data found by human intervention (such as would occur during approval, etc.) necessary to correctly re-process the failed events. The whole of these secondary tables preferably should:

BNSDOCID: <WO__9523373A1_I_>

1) include storage for information created during the billing process;

2) include storage for any new records appended to the customer or reference data; and

3) include an identifier for the run on which it was created.

All new information calculated during the billing process is tied through an artificial key to the original primary customer charge information.

In addition, a table containing all accounts comprising a full batch run is provided with a status flag indicating the current state of the account. The valid states are either unapproved or approved. An unapproved account is available for processing by a billing process. An approved account has been determined, either by manual intervention or automatic determination such as through some form of artificial intelligence, for example, to have been completely and correctly processed. Further defining the account table is a pair of batching tables which provide descriptions of the parent batch (typically the full batch run) and a run ID number. Like the accounts, state information is stored for the batches. The valid states can be one of pending, active or complete. A pending batch is available for processing by a billing process. An active batch is one that is currently being handled by a billing process. A complete batch is one which has completed a billing process.

Each account is associated to the batching tables by maintaining its maximum run ID. In other words, every time an account is run, it stores an initial value for the run ID or the most current run ID. The maximum run ID is used to determine what pieces of data are the most current associated with each customer. Using the maximum run ID in this manner eliminates the need to delete old customer information when an account is re-processed. The data identified by a run ID equal to the

BNSDOCID: <WO__9523373A1_I_>

accounts maximum run ID is the only correct data available for that customer.

An initial batch is submitted to the MTBP prior to the first request for processing. An external approval process and automatic determination by the billing process are used to handle re-submission of batches to the MTBP. A human operator is responsible for determining the completeness and correctness of a batch of accounts. Any accounts within the batch that are determined to be incorrect are re-submitted as a new batch comprised of only those accounts in error. Any accounts that contain one or more fatal processing errors or can otherwise be determined to have incorrectly processed are automatically re-batched for re-processing.

As an additional process, approval/revenue reporting and invoicing can be removed from the billing process. These processes can be placed in an external process. The external reporting mechanisms are capable of generation of their respective reports and invoices at any point after the completion of a billing process and can be created on any subset of the processed batches.

The overall operation of a discrete billing function with a no-reset function may occur within the multi-threaded batch processor. As such, the process as depicted in Fig. 5 would be substantially the same. One embodiment of the present invention for providing a no-reset function is represented in the flowchart of Fig. 8. In Figs. 8 and 9, numbers in common with Fig. 5 represent like process steps. In Fig. 8, when the MTBP reaches step 518 and it is determined that the segment did not complete without error, the MTBP does not kill all remaining segments as in step 526. Rather, in step 800 the segment data is re-initialized using the secondary tables associated with the customers in which the data that could have been changed is stored. At this point, a request to fork a new segment is issued by the MTBP in step 802. The process of re-initializing

and re-forking preferably requires human intervention to resolve the error requiring the re-processing. However, automatic re-processing using some form of artificial intelligence could also be possible. For example, it may be desirable to attempt to process a particular segment twice before involving a human operator. Therefore, the human operator has to resolve the problem only if the error is due to some form of data error or equipment problem rather than simply a bad data read or the like.

In another embodiment, as depicted in Fig. 9, only the failed discrete events, which may be customer accounts for example, are re-processed. In this embodiment, if a segment fails as determined in step 518, in step 900, it is determined which of the discrete events were processed without failure and which were not. Again, this process could be done automatically by comparing the discrete event to some range of acceptable values, or could be done through an operator. Once it is determined which customer accounts should be reprocessed, a new segment is formed in step 902 comprising the failed customer accounts. This new segment is then forked in step 904 and control passes back to the 516-520 loop.

Having described the preferred embodiments of the invention, it will be appreciated by those skilled in the art that there exist numerous alternatives and equivalents which do not depart from the scope or spirit of the invention. Accordingly, it is intended that the scope of the present invention be limited only by the appended claims and not by the above description of the above preferred embodiments.

BNSDOCID: <WO__9523373A1_I_>

CLAIMS

What is claimed is:

1. A method of processing a plurality of discrete events, each discrete event comprising a plurality of independent sub-events, said method comprising the steps of:

distributing each discrete event into one of a plurality of segments, each segment comprising a sequence of at least one discrete event to be processed;

initiating each of said plurality of segments to execute concurrently on said at least one processor;

for each segment, processing each discrete event contained within said segment sequentially;

for each discrete event, processing each independent sub-event of said discrete event sequentially and then storing the results of said processing;

monitoring each of said segments to detect failures;

deactivating each segment for which a failure is detected;

re-initializing each failed segment; and

re-initiating each failed segment.

2. The method of claim 1 wherein said discrete event is a customer account and wherein said step of processing each discrete event comprises determining billing information for each customer account.

3. The method of claim 1 wherein said discrete event is a customer account and wherein said independent sub-events comprise customer information and customer calls.

4. The method of claim 1 wherein said discrete event is a customer account, wherein said step of processing each discrete event comprises determining billing information and wherein billing information for each

customer account is stored immediately after each customer account is processed.

5. The method of claim 1 wherein said discrete event is a customer account and wherein said step of processing each discrete event comprises determining billing information and further comprising the step of generating an invoice for each customer account immediately after processing each customer account.

6. The method of claim 1 wherein said step of processing is performed using a symmetrical multiprocessing system.

7. The method of claim 1 wherein said step of processing is performed using a massively parallel processing system.

8. The method of claim 1 wherein said step of processing is performed using a loosely coupled distributed processing system.

9. A method of processing a plurality of discrete events, each discrete event comprising a plurality of independent sub events, said method comprising the steps of:

distributing each discrete event into one of a plurality of segments, each segment comprising a sequence of at least one discrete event to be processed;

initiating each of said plurality of segments on said at least one processor;

for each segment, processing each discrete event contained within said segment sequentially;

for each discrete event, processing each independent sub-event of said discrete event sequentially and then storing the results of said processing;

monitoring each segment to detect failures in processing the discrete events contained within said segment;

deactivating each segment for which a failure is detected;

BNSDOCID: <WO 9523373A1_I_>

creating a new segment containing at least one unprocessed discrete event from each failed segment; and

initiating each new segment.

10. The method of claim 9 wherein a plurality of new segments are created, each of said new segments containing at least one unprocessed discrete event.

11. The method of claim 9 wherein said discrete event is a customer account and wherein said step of processing each discrete event comprises determining billing information for each customer account.

12. The method of claim 9 wherein said discrete event is a customer account and wherein said independent sub-events comprise customer information and customer calls.

13. The method of claim 9 wherein said step of processing is performed using a symmetrical multiprocessing system.

14. The method of claim 9 wherein said step of processing is performed using a massively parallel processing system.

15. The method of claim 9 wherein said step of processing is performed using a loosely coupled distributed processing system.

16. A computer system for processing a plurality of discrete events, each discrete event comprising a plurality of independent sub-events, said system comprising:

distributing means for distributing each discrete event into one of a plurality of segments, each segment comprising a sequence of at least one discrete event to be processed;

initiating means for initiating each of said plurality of segments to execute concurrently on at least one processor;

said at least one processor for processing each discrete event sequentially within each segment and for processing each independent sub-event of each discrete event sequentially within each discrete event;

BNSDOCID: <WO 9523373A1 I >

memory means for storing each processed discrete event;

monitoring means for monitoring each segment to detect failures in processing the discrete events contained within each segment;

deactivating means for deactivating each segment for which a failure is detected;

initializing means for creating a new segment containing at least one unprocessed discrete event from each failed segment; and

initiating means for initiating each new segment.

17. The system of claim 16 wherein a plurality of new segments are created, each of said new segments containing at least one unprocessed discrete event.

18. The system of claim 16 wherein said discrete event is a customer account and wherein said step of processing each discrete event comprises determining billing information for each customer account.

19. The system of claim 16 wherein said discrete event is a customer account and wherein said independent sub events comprise customer information and customer calls.

20. The system of claim 16 wherein said discrete event is a customer account, wherein said processing means determines billing information and wherein billing information for each customer account is stored in said memory means immediately after each customer account is processed.

21. The system of claim 16 wherein said discrete event is a customer account, wherein said processing means determines billing information and further comprising the step of generating an invoice for each customer account immediately after processing each customer account.

22. The system of claim 16 wherein said at least one processor comprises a symmetrical multiprocessing system.

23. The system of claim 16 wherein said at least one processor comprises a massively parallel processing system.

24. The method of claim 16 wherein said at least one processor comprises a loosely coupled distributed processing system.

25. A computer system for processing a plurality of discrete events, each discrete event comprising a plurality of independent sub events, said system comprising:

distributing means for distributing each discrete event into one of a plurality of segments, each segment comprising a sequence of at least one discrete event to be processed;

first initiating means for initiating said plurality of segments on at least one processor;

at least one processor for processing each discrete event sequentially within each segment and for processing each independent sub event of each discrete event sequentially within each discrete event;

memory means for storing each processed discrete event;

monitoring means for monitoring each of said segments to detect failures;

deactivating means for deactivating each segment for which a failure is detected;

segment creating means for creating a new segment from the failed segment comprising the unprocessed discrete events from the failed segment; and

second initiating means for initiating the new segment.

26. The system of claim 25 wherein said discrete event is a customer account and wherein at least one processor determines billing information for each customer account.

27. The system of claim 25 wherein said discrete event is a customer account and wherein said independent sub events comprise customer information and customer calls.

BNSDOCID: <WO__9523373A1_I_>

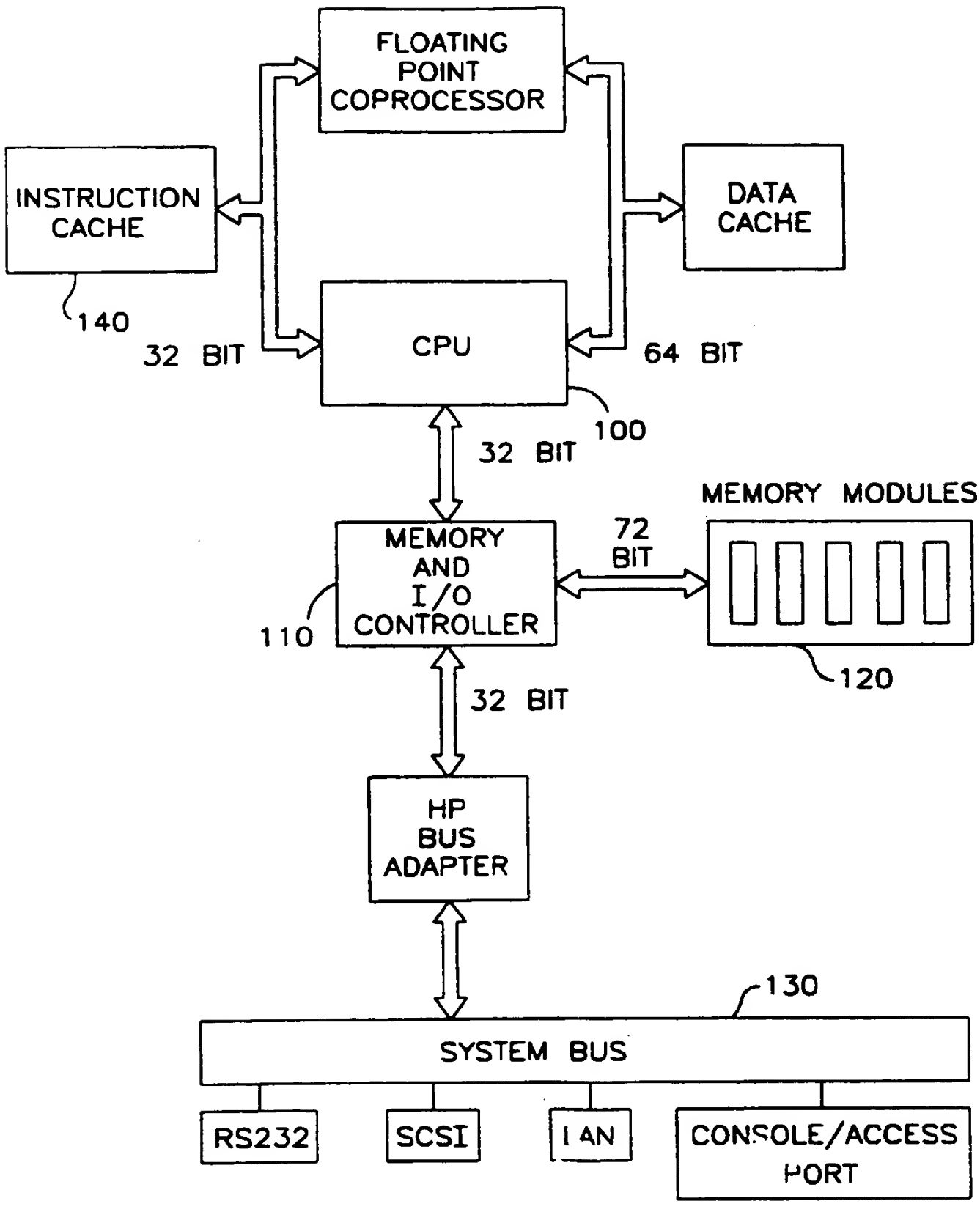


FIG. 1

BNSDOCID: <WO__ 9523373A1_I_>

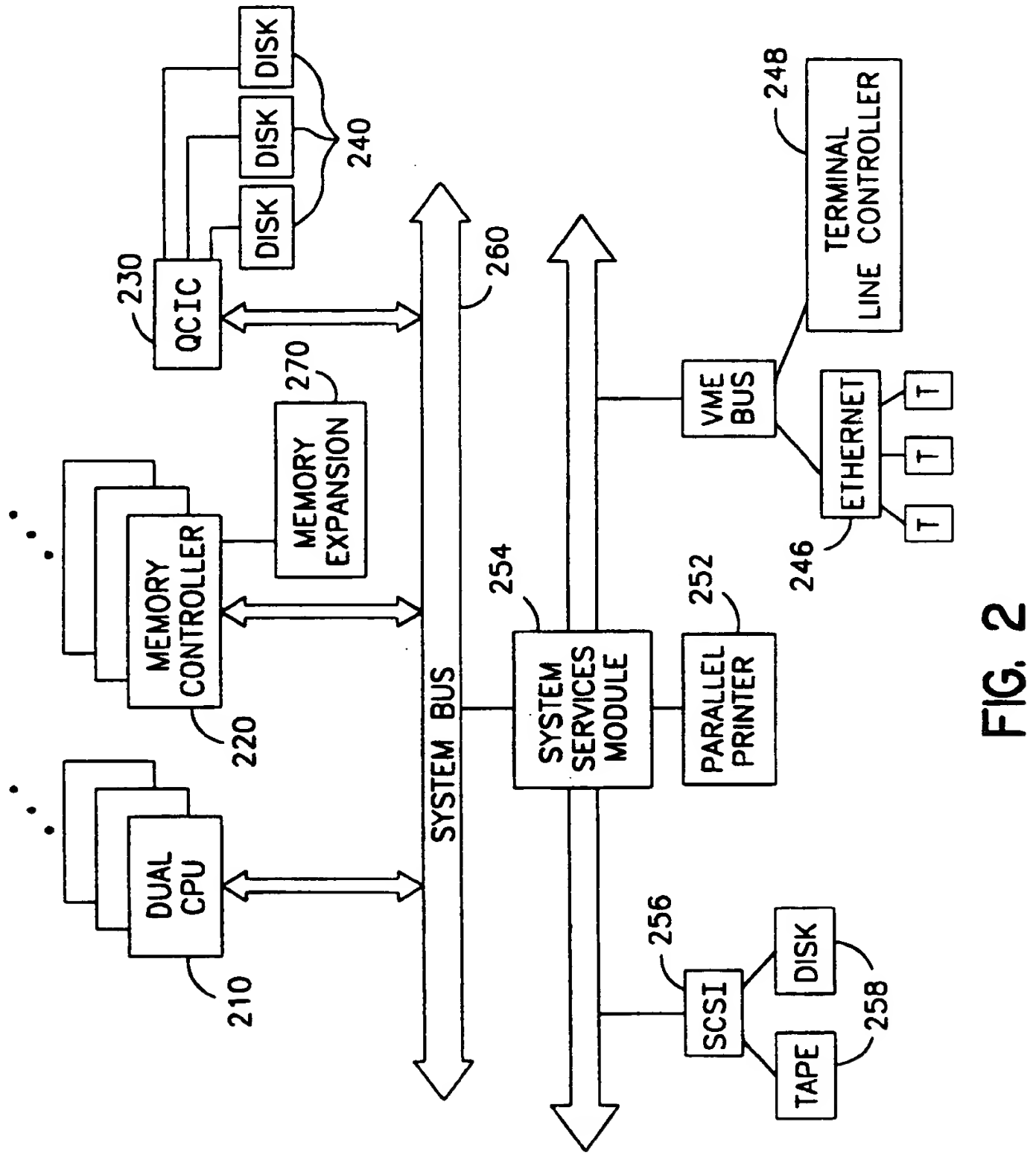


FIG. 2

BNSDOCID: <WO___9523373A1_I_>

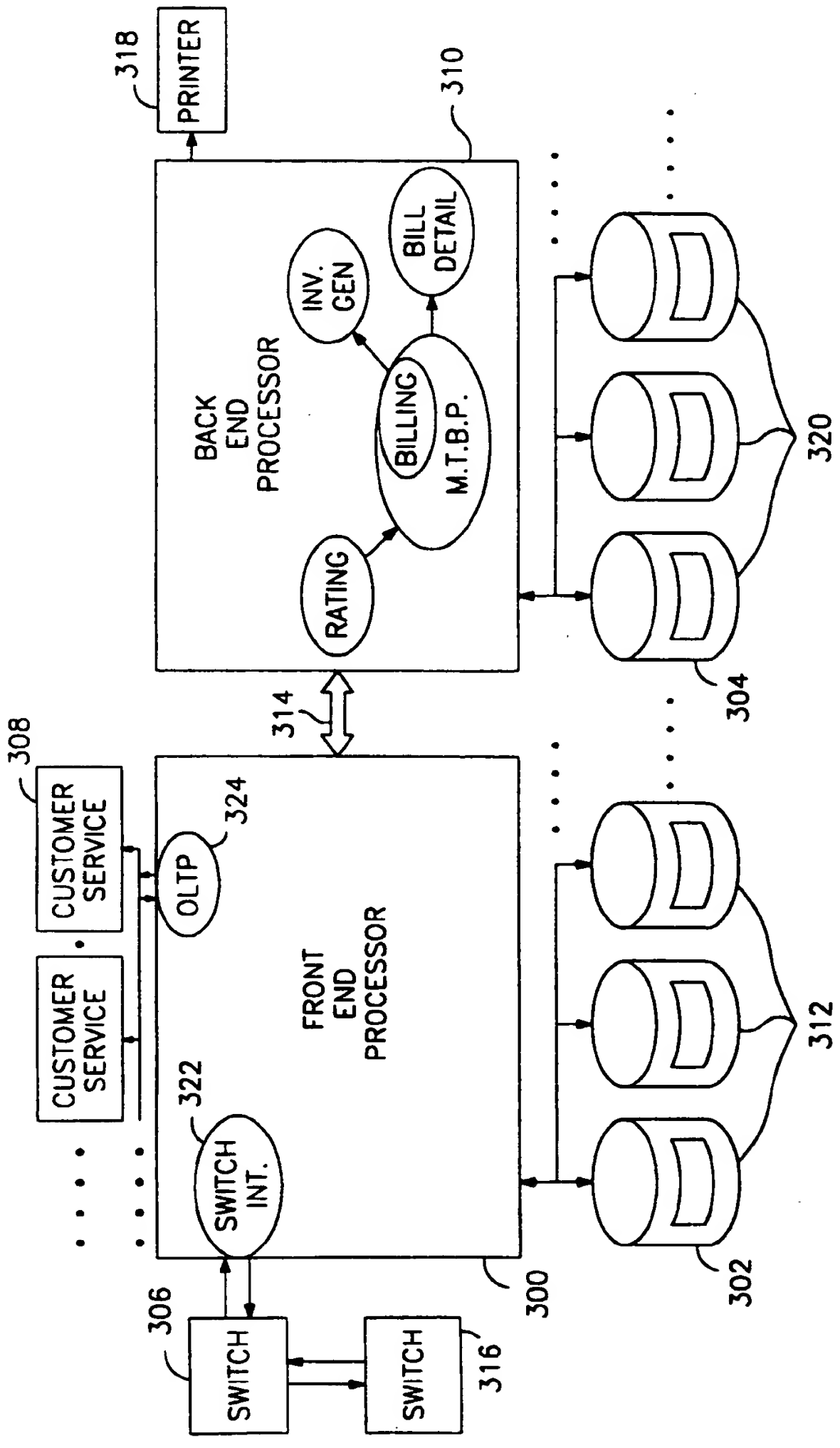


FIG. 3

BNSDOCID: <WO___9523373A1_I_>

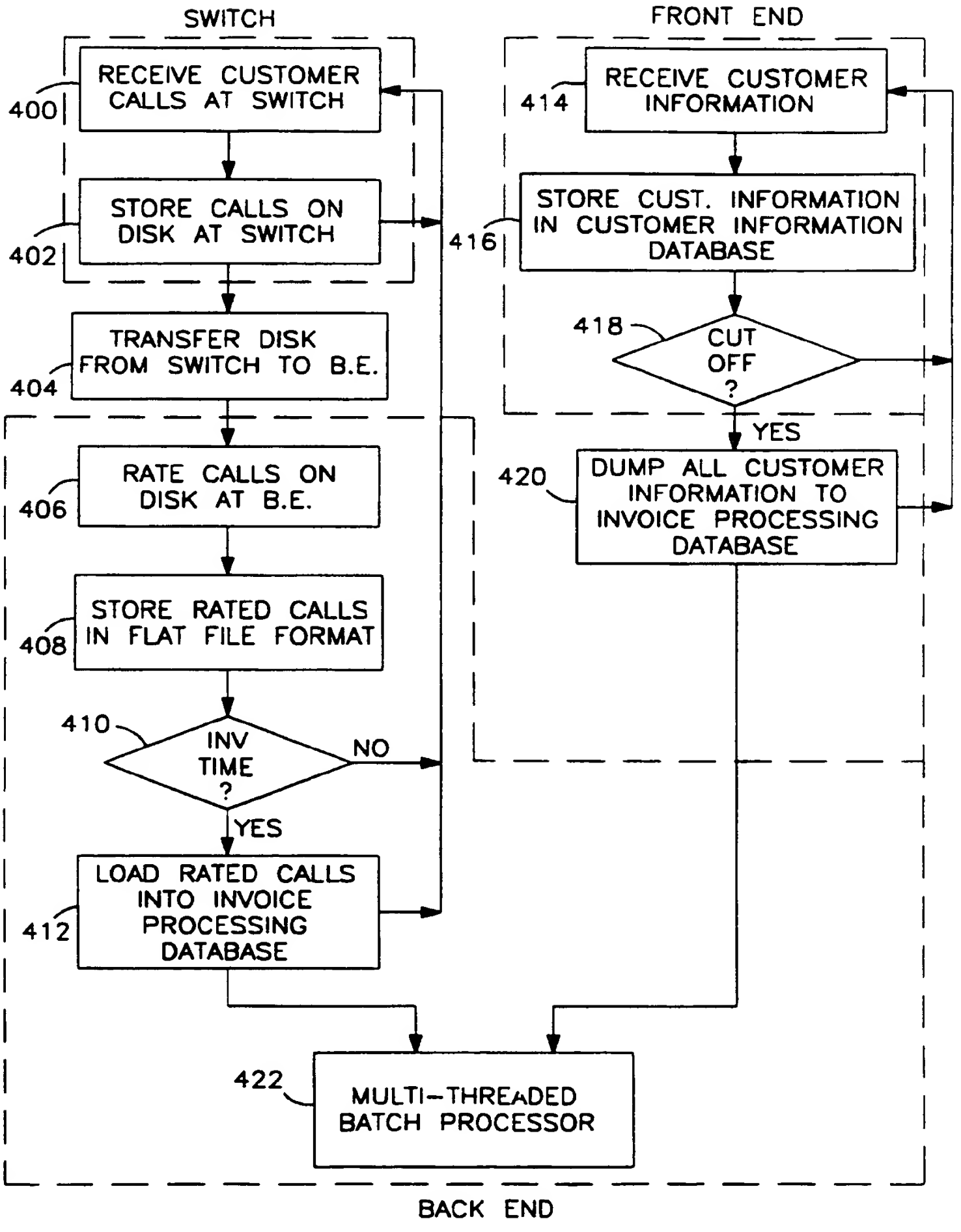


FIG. 4

BNSDOCID: <WO___9523373A1_I_>

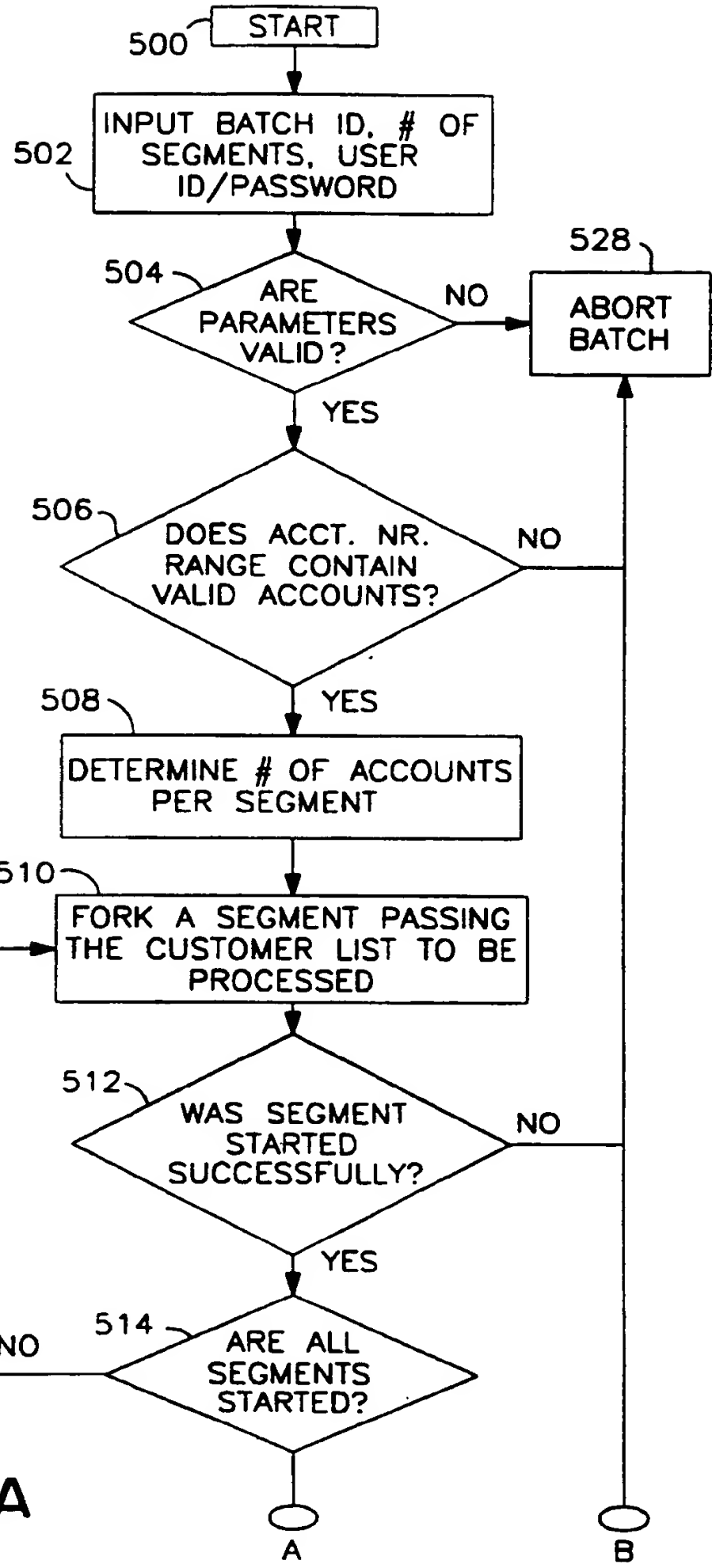


FIG. 5A

BNSDOCID: <WO 9523373A1_I >

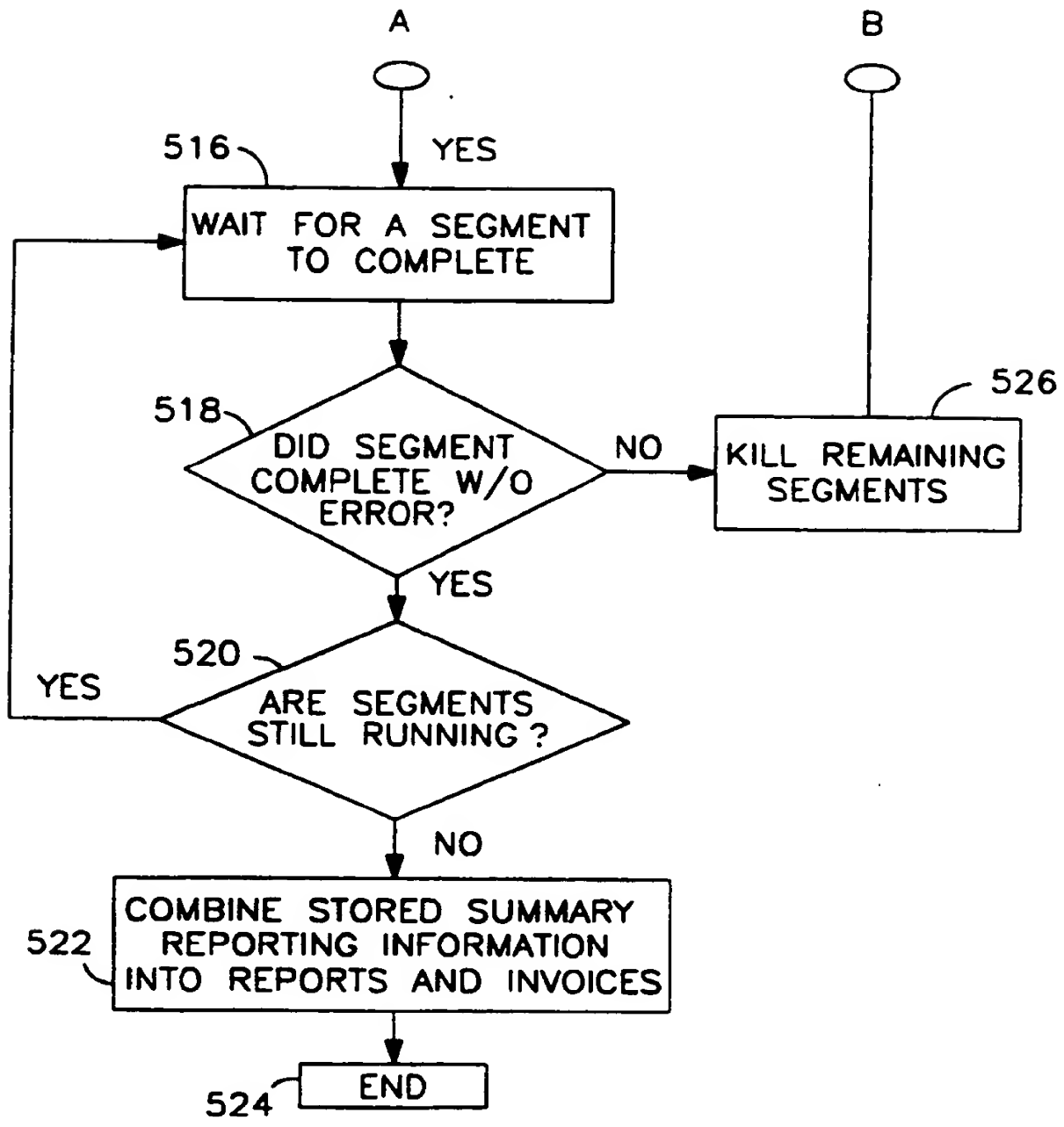


FIG. 5B

BNSDOCID: <WO___9523373A1_I_>

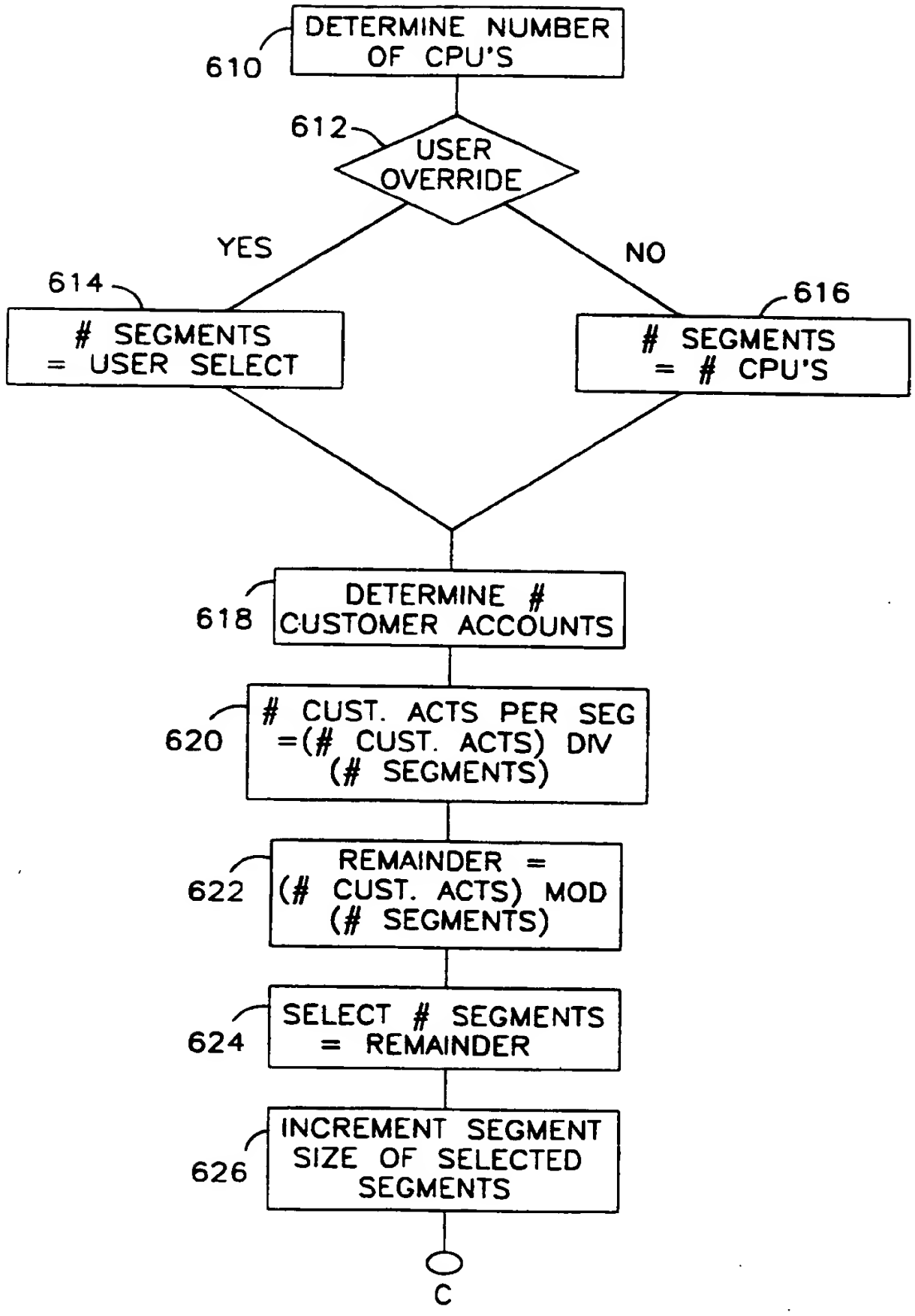
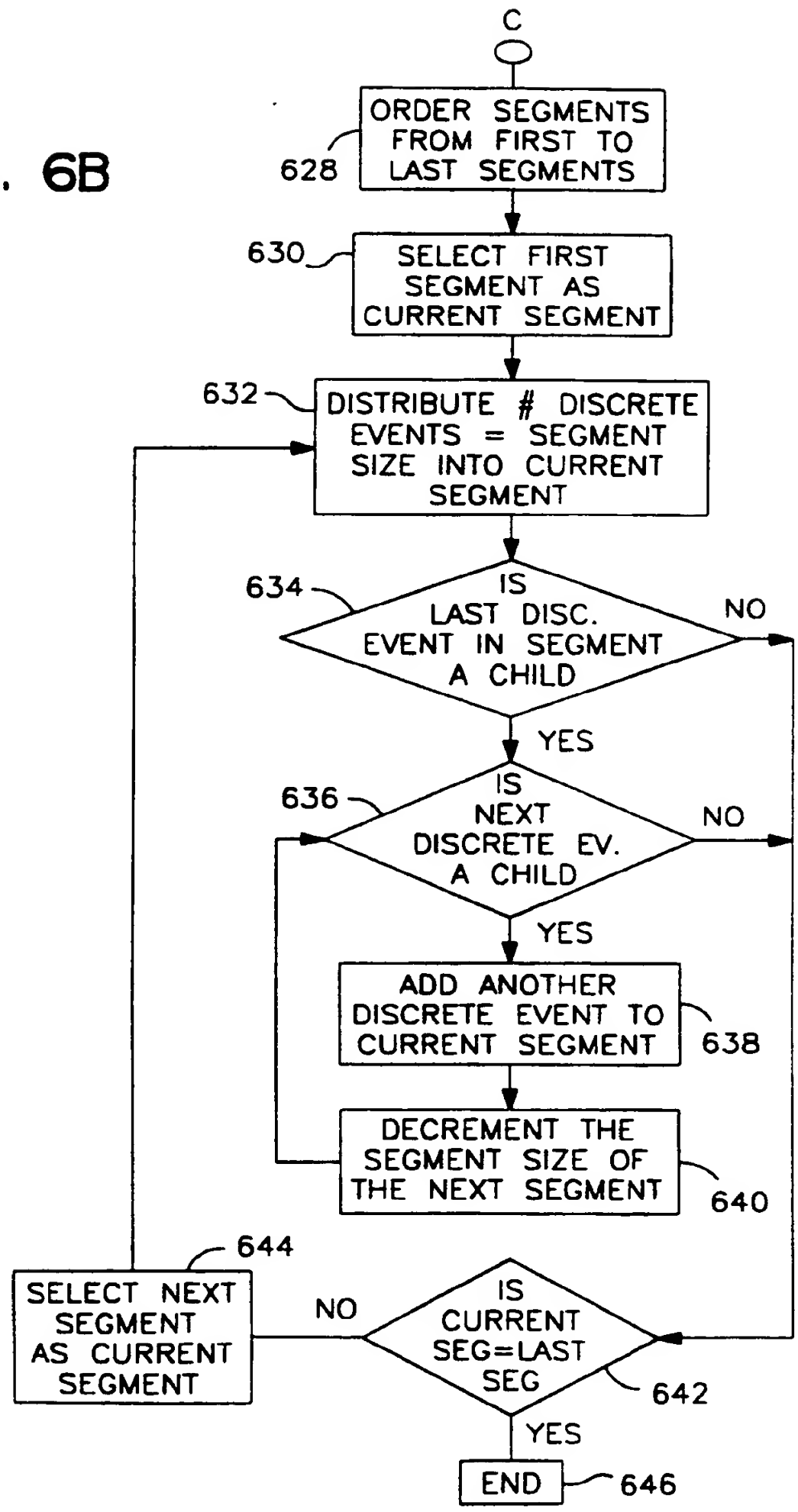


FIG. 6A

BNSDOCID: <WO___9523373A1_I_>

FIG. 6B



BNSDOCID: <WO___9523373A1_I_>

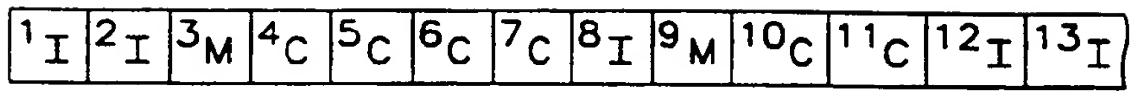


FIG. 7

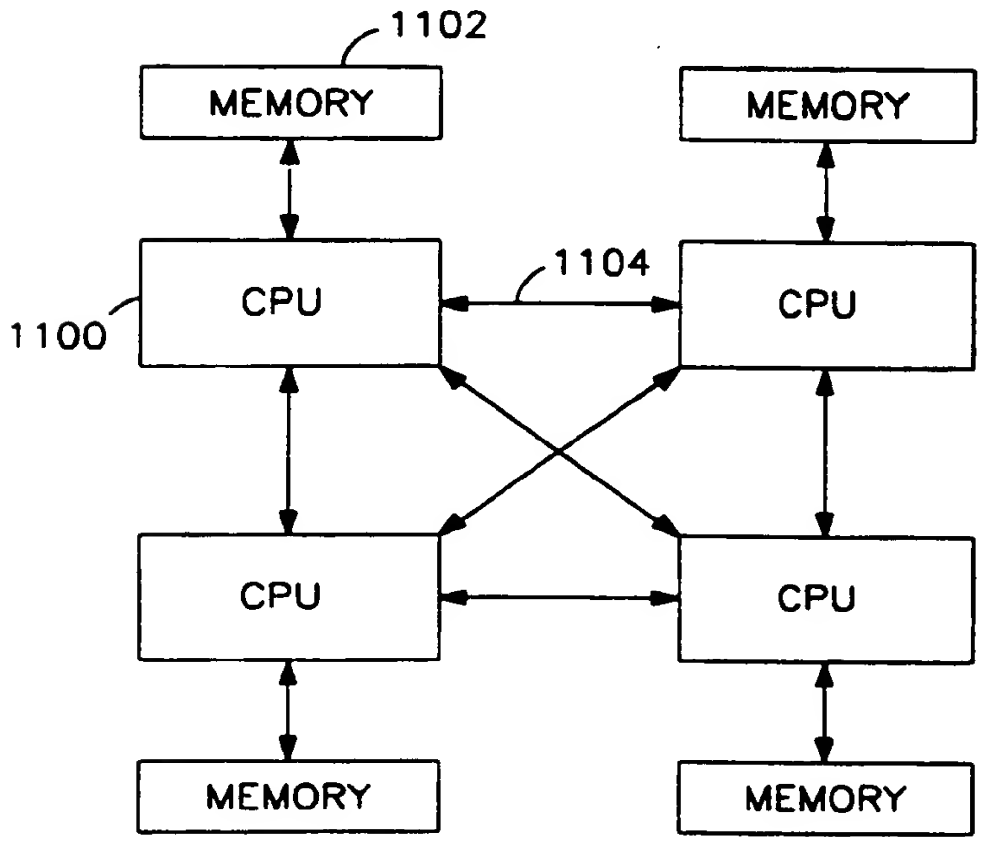


FIG. 11

BNSDOCID: <WO__9523373A1_I_>

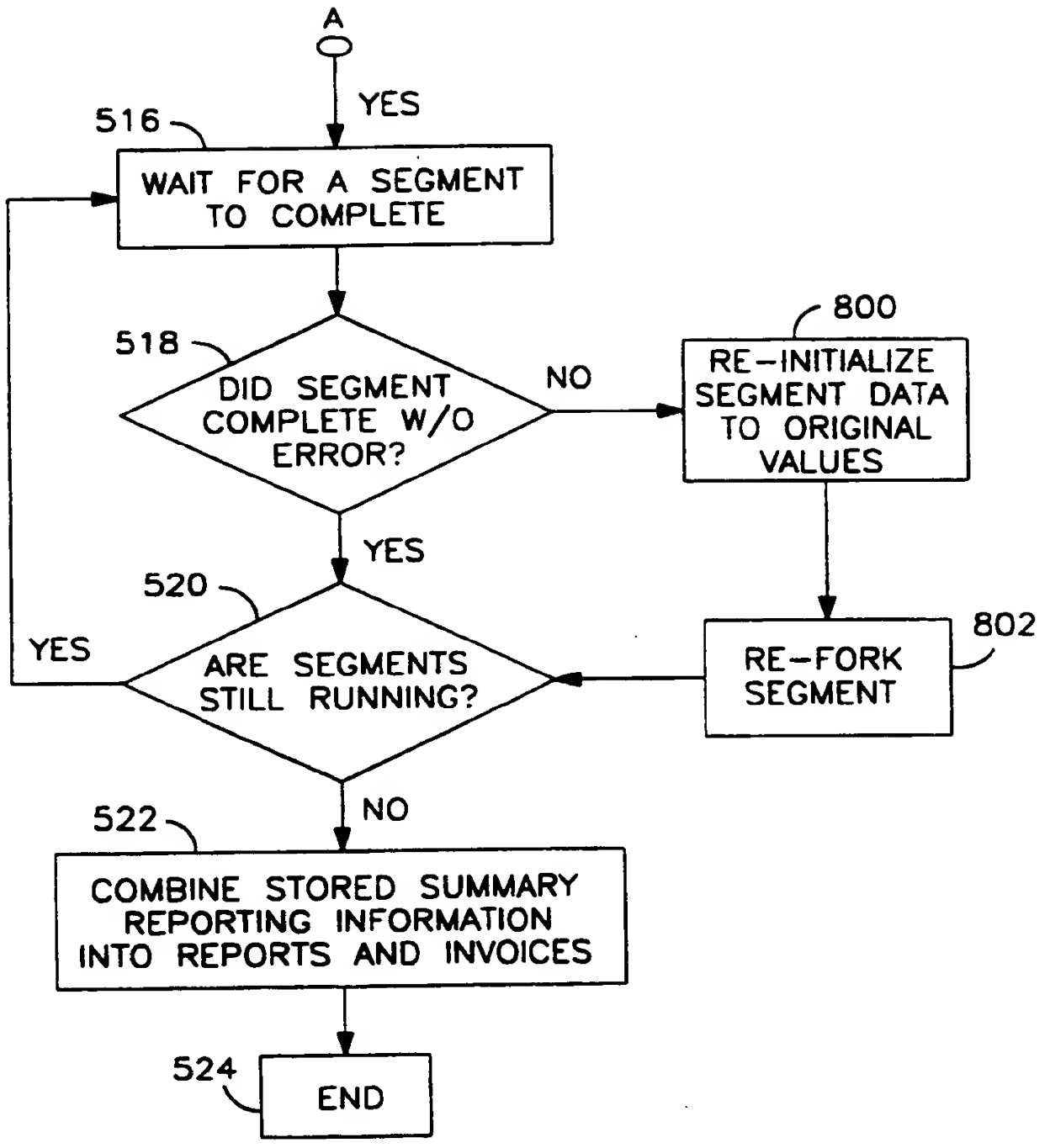


FIG. 8

BNSDOCID: <WO___9523373A1_I_>

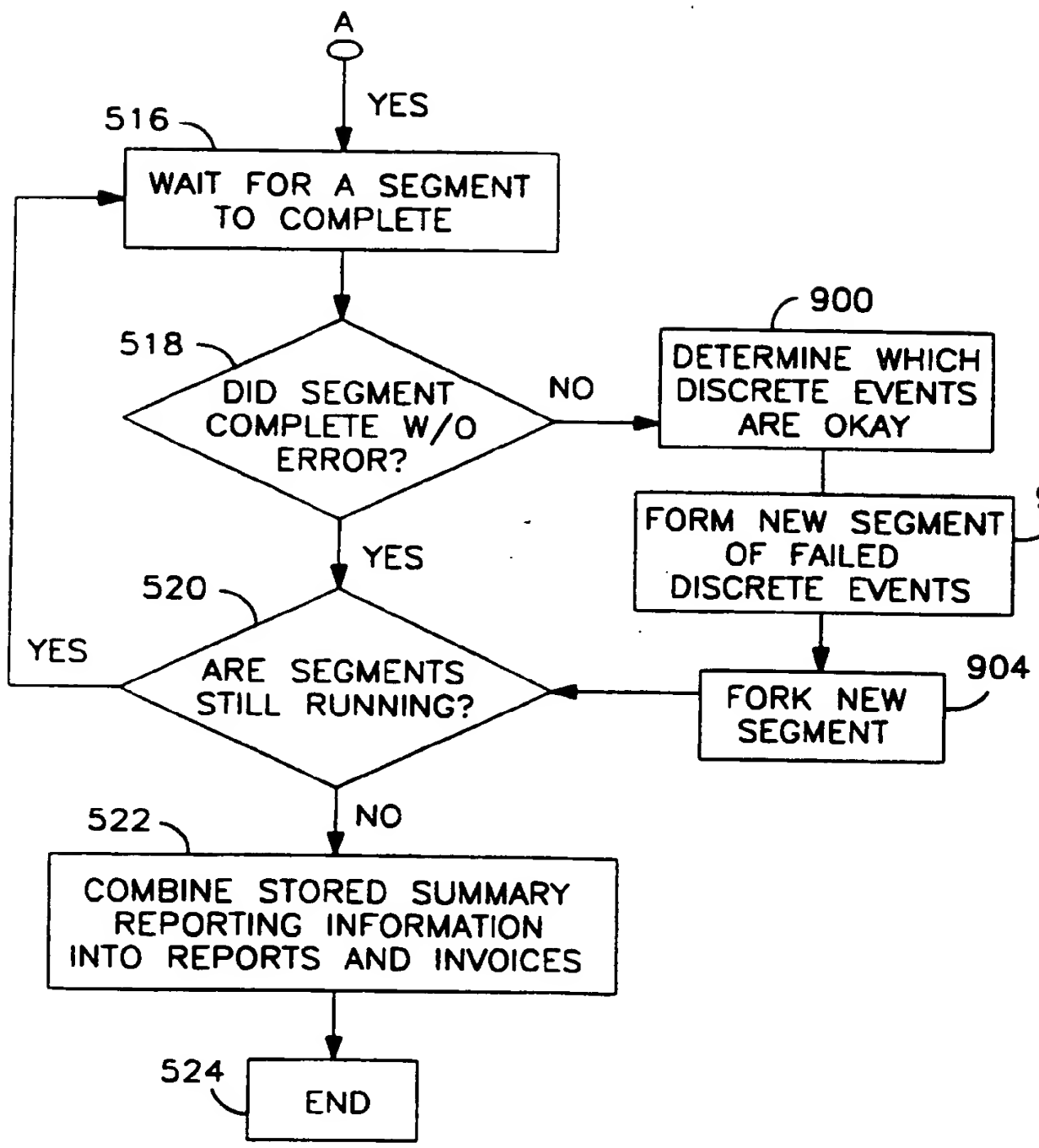


FIG. 9

BNSDOCID: <WO__ 9523373A1_I_>

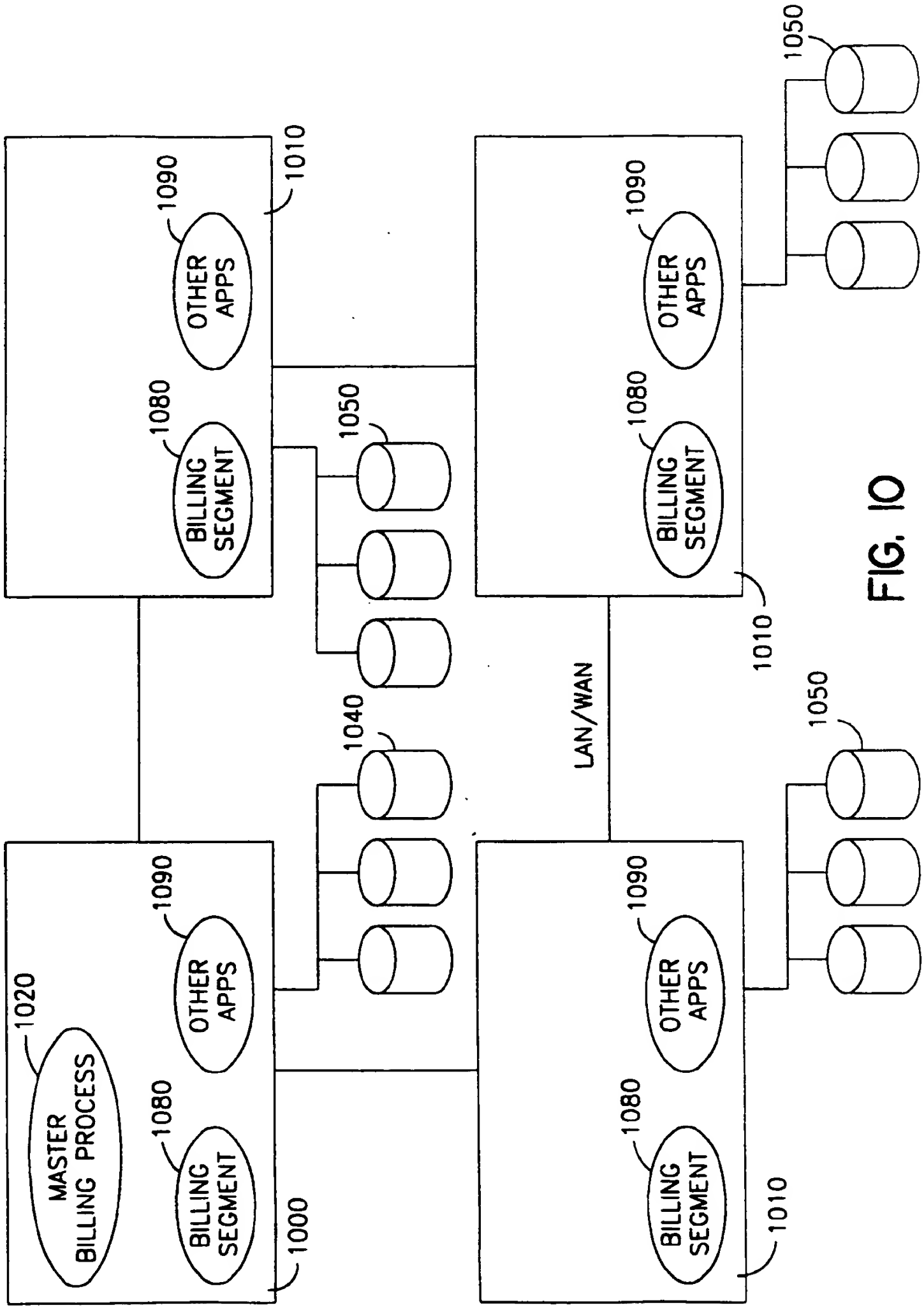


FIG. 10

BNSDOCID: <WO___9523373A1_I_>

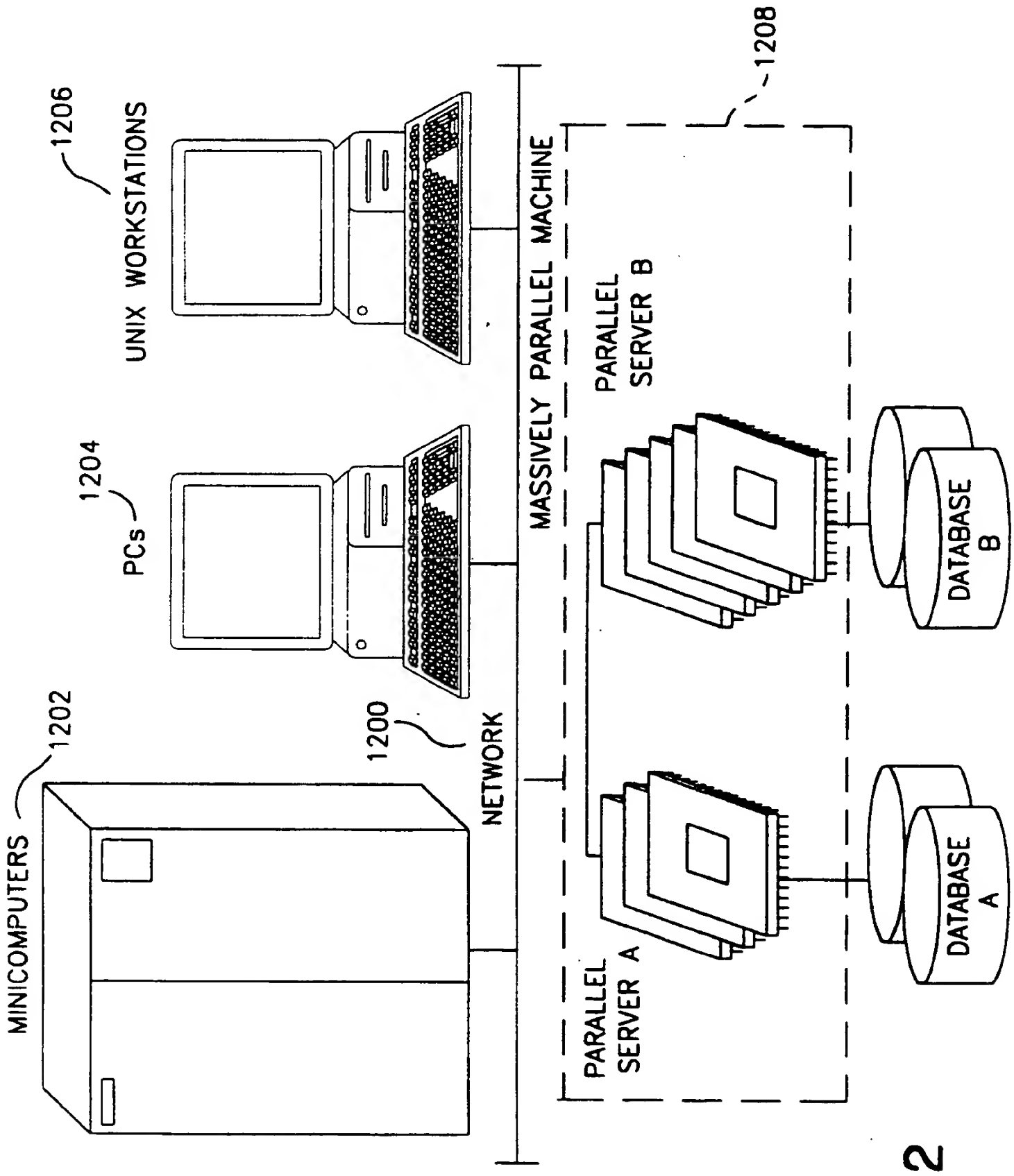


FIG. 12

BNSDOCID: <WO_ 9523373A1_I_>